NPS-CS-00-006

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Management System for Heterogeneous Networks Final Report
Volume I: Project Summary and Papers (Part A)

by

Cynthia E. Irvine H. J. Siegel Viktor Prasanna
Debra Hensgen Timothy Levin

14 April 2000

Approved for public release; distribution is unlimited.

Prepared for: Defense Advanced Research Project Agency
Information Technology Organization

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM Richard H. Wells, USNR
Superintendent

R. Elster
Provost

This report was prepared for the Naval Postgraduate School Center for Information Systems Security (INFOSEC) Studies and Research (NPS CISR) at the Naval Postgraduate School, as part of a project funded under the Defense Advanced Research Projects Agency/Information Technology Organization grant under the Quorum program.

This report was prepared by:

Cynthia E. Irvine

Cynthia E. Irvine
Assistant Professor
Naval Postgraduate School

Timothy Le

Timothy Levin
Senior Research Associate
Anteon Corporation

Reviewed by:

Neil C. Rowe

Neil C. Rowe
Associate Professor
Department of Computer Science

DC Boger

D. C. Boger, Chair
Department of Computer Science

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE

Form approved

OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 14 April 2000	**3. REPORT TYPE AND DATES COVERED** Final Report

4. TITLE AND SUBTITLE
Management System for Heterogeneous Networks Final Report, Volume I: Project Summary and Papers

5. FUNDING

MIPR No. 00-E583

6. AUTHOR(S)

Cynthia E. Irvine, H.J. Siegel, Viktor Prasanna, Debra Hensgen and Timothy Levin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School Center for Information Systems Security Studies and Research (NPS CISR)
Naval Postgraduate School, 833 Dyer Road, Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

NPS-CS-00-006

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DARPA/ITO
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words.)

The goal of the MSHN Project was to explore the application of adaptive and heuristic matching and scheduling techniques, and modern distributed security methods, to a distributed heterogeneous resource management system (RMS) which allows system resources to be accessed by both MSHN-controlled and external applications. This document provides both a high-level overview of the MSHN technical program and a reference guide to the MSHN research papers constituting Appendix A.

14. SUBJECT TERMS

resource management system, distributed systems, matching and scheduling

15. NUMBER OF PAGES 787

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIRIED	**20. LIMITATION OF ABSTRACT** UNLIMITED

Management System for Heterogeneous Networks
Final Report
Volume I: Project Summary and Papers

Naval Postgraduate School Technical Report NPS-CS-00-06

Cynthia E. Irvine
Naval Postgraduate School

H. J. Siegel
Purdue University

Viktor Prasanna
University of Southern California

Debra Hensgen
Formerly with
Naval Postgraduate School

Tim Levin
Anteon Corporation

April 14, 2000

This page is intentionally blank

1 Introduction

The Management System for Heterogeneous Networks (MSHN) project is part of the DARPA/ITO Quorum program. Quorum's goal is to develop technologies to allow mission-critical defense applications to achieve survivable, predictable, and controllable quality of service on a globally managed pool of distributed resources.

The goal of the MSHN Project is to explore the application of adaptive and heuristic matching and scheduling techniques, and modern distributed security methods, to a distributed heterogeneous resource management system (RMS) which allows system resources to be accessed by both MSHN-controlled and external applications. To validate our research and engineering assumptions, a prototype version of MSHN has been developed and demonstrated.

A complete description of the MSHN technical program is found in the research papers which constitute Appendix A. The remainder of this document provides both a high-level overview of the MSHN technical program and a reference guide to the research papers.

The MSHN Project began in 1997, under the direction of Dr. Debra Hensgen. In the fall of 1999, Dr. Cynthia Irvine took on oversight for MSHN. The primary project contract concluded on March 31, 2000. These are the MSHN investigators:

- **Principal Investigators**
  - Dr. Cynthia Irvine, Naval Postgraduate School
  - Richard Freund, Noemix, Inc.
- **Investigators**
  - Dr. Viktor Prasanna, University of Southern California
  - Dr. H.J. Siegel, Purdue University
- **Past Principal Investigators**
  - Dr. Debra Hensgen, formerly with Naval Postgraduate School
  - Dr. Taylor Kidd, formerly with Naval Postgraduate School

2 Architecture

The MSHN design embodies a peer-to-peer architecture [23] composed of the following components:

- Client Library (wrapping each application under MSHN's control)
- Scheduling Advisor (hierarchically replicated)
- Resource Requirements Database (hierarchically replicated)
- Resource Status Server (hierarchically replicated)
- MSHN Daemon (one for each computing resource)
- Application Emulator (at least one for each computing resource)

These components can execute on the same physical machine or can be distributed to reside on separate, heterogeneous machines. The Common Object Request Broker Architecture (CORBA) provides communication between components. Communication security between the MSHN components is provided by the MSHN Security Architecture [57] [56] [22].

The MSHN architecture supports the simultaneous execution of many different client applications, supporting both new and previously encountered applications. MSHN does not assume complete control of its managed resources; rather it allows both MSHN and non-MSHN (viz, non-wrapped) applications to access system resources. Because resources are continuously monitored, external and legacy applications that are not wrapped by the Client Library are accounted for indirectly by their interaction with the system resources.

2.1 Other Architectures

For a comparison with other resource management and heterogeneous computing architectures, see [42]. This work provides background for various aspects of our MSHN work by summarizing relevant papers from a variety of research projects. We include (1) a broad overview of heterogeneous computing (HC); (2) several case studies that give more specific details of applications executing on HC systems; (3) a sampling of current HC tools and environments; (4) methods of classifying HC systems; and (5) techniques for benchmarking machines, techniques for profiling tasks, and schemes that use the information regarding machines and tasks to derive a mapping of the tasks onto the machines.

2.2 Basic System Functions and Attributes

When viewed as a black box, the MSHN system interacts with two actors: applications and resources. MSHN's primary job is assigning and reassigning resources to applications. Included in that functionality is the discovery of resources and the monitoring of both availability of those resources and requirements of the applications that make use of those resources.

2.2.1 Scheduling Advisor Functions

The primary responsibility of the Scheduling Advisor is to determine the best assignment of resources to a set of tasks based on the optimization of a global metric. The Scheduling Advisor depends on the Resource Requirements Database and the Resource Status Server in order to identify an operating point that optimizes the global metric. It responds to scheduling and resource assignment requests from the Client Library. When appropriate, the Scheduling Advisor requests application adaptations via the Client Library. The Scheduling Advisor is also responsible for establishing thresholds to trigger callbacks to the Resource Status Server and Resource Requirements Database (see details below).

2.2.2 Client Library Functions

The Client Library is intended to be linked with both adaptive and non-adaptive applications. It provides the application with a transparent interface to all of the other MSHN components. The Client Library intercepts system calls to collect resource usage and status information, which it forwards to the Resource Requirements Database and the Resource Status Server. The Client Library also intercepts calls that initiate new processes (such as exec()) and consults the Scheduling Advisor for the best place to start that process. It requests execution of applications based on advice from the Scheduling Advisor. Similarly, when notified by the Scheduling Advisor via callbacks, the Client Library can trigger changes to adaptive applications, including the Application Emulator.

2.2.3 Resource Status Server Functions

The role of the Resource Status Server is to maintain a repository of the three types of information about the resources available to MSHN: relatively static (long-term), moderately dynamic (medium-term), and highly dynamic (short-term) information. The Resource Status Server is updated with current data via the Client Library. The Resource Status Server responds to Scheduling Advisor requests with estimates of currently available resources. The Scheduling Advisor sets up callbacks with the Resource Status Server based on resource availability thresholds and Client Library update frequency requirements.

2.2.4 Resource Requirements Database Functions

The Resource Requirements Database is intended to be a repository of information pertaining to the resource usage of applications. The Resource Requirements Database provides this information to the Scheduling Advisor, and it is updated by the Client Library. Callbacks to the Scheduling Advisor are based on either the occurrence of a threshold violation or update frequency requirements.

2.2.5 Daemon (D) Functions

The MSHN Daemon runs on all compute resources available for use by the Scheduling Advisor. It's sole purpose is to start applications as requested by the Client Library.

2.2.6 Application Emulator

The Application Emulator serves two purposes. The first is to simulate applications (that statistically have the same resource usage footprint as the real applications) without the overhead and uncertainty of actually installing, maintaining and running that particular application. The second purpose is to be a resource availability monitor in the absence of any other MSHN-wrapped applications. The daemon starts one instance of the Application Emulator by default at startup. For the purposes of the MSHN demonstration, the Application Emulator functions are performed by a version of the MSHN Daemon.

3 Mapping Algorithms

The mapping (matching and scheduling) research we have conducted was in support of the MSHN Scheduling Advisor. The Scheduling Advisor will include a "toolbox" of mapping techniques from which it can select the most appropriate to use for any given heterogeneous computing and application environment.

3.1 Unified Mapping Framework

We have developed a unified mapping framework for heterogeneous computing systems [3]. Our framework considers multiple types of resources such as compute resources, network resources, I/O devices, and data repositories, such that mapping decisions are based on all the resource requirements. Using our framework, we formulated and studied two novel mapping problems:

- Mapping with advance reservation and data replication

- Mapping with resource co-allocation requirements

In the first problem, we considered the emerging concept of advance reservations where system resources can be reserved in advance for specific time intervals. We assumed that applications with various resource requirements are submitted from participant sites. Each application was assumed to consist of several tasks and was represented by a directed acyclic graph (DAG). The resource requirements were specified at the task level. A task's input data can be data items from its predecessors and/or data sets from data repositories. Input data sets can be accessed from one or more data repositories. A task is ready for execution if all its predecessors have completed, and it has received all the input data needed for its execution. Sources of input data and the execution times of the tasks on various machines along with their availability were considered simultaneously to minimize the overall completion time.

We have developed several heuristic algorithms to solve the above problem. These results are published in [1]. Although we considered multiple resource requirements, tasks were not required to access different types of resources simultaneously.

In the second problem, we considered mapping a set of applications in a heterogeneous computing (HC) system where application tasks require concurrent access to multiple resources of different types. In general, this problem is the resource co-allocation problem. The co-allocation problem can be defined as the problem of simultaneously allocating multiple resources of different types to applications in order to meet specific performance requirements.

We have developed a general framework for mapping with resource co-allocation in HC systems. The framework defined the system and application models and formulated the co-allocation problem. Two graphs were used to represent applications: a directed acyclic graph and a "compatibility graph." The DAG representation is given initially and it stays unchanged throughout the mapping process while the compatibility graph is updated during the mapping process. In classical mapping problems, only DAGs are used to represent the precedence constraints among tasks. In our framework, the co-allocation requirements add another type of constraint among the tasks: resource sharing constraint which is captured by the compatibility graph. Tasks that share one or more resources cannot be executed concurrently due to resource sharing constraints even if they have no precedence constraints among them. Known mapping algorithms for the classical DAG scheduling problem cannot be directly used for the above problem since they consider the precedence constraints only. We have developed heuristic algorithms that can be used with different allocation techniques to efficiently solve the co-allocation problem defined by our framework.

In our approach, multiple DAGs of different applications are combined into a single DAG. All tasks that have satisfied the precedence constraints are ready for allocation provided they have no resource sharing constraints. Using the compatibility graph, we select tasks that can be executed concurrently. This is achieved by finding maximal independent sets in the compatibility graph. These results appear in [2].

3.2 Mapping Heuristics

We have studied heuristics for mapping (viz, scheduling and matching) communicating subtasks to machines in a variety of situations. A genetic algorithm method for static (off-line) mapping of communicating subtasks of a task in a heterogeneous computing (HC) environment is presented in [55]. A way to select from precomputed static mappings, using on-line real-time feedback for automatic target recognition problems is given in [35]. In [41] [38], we describe a hybrid remapper that improves a statically obtained initial mapping by using on-line feedback of run-time execution times of communicating subtasks and machine ready times. A theoretical stochastic model for the mapping of communicating subtasks of a task is presented in [52]. This model is used to show the worth of a greedy approach for mapping heuristics.

We have also considered the case where the tasks to be mapped to machines are independent. Eleven different static mapping heuristics are compared in [15] [14] under several different situations that could occur in a heterogeneous computing environment. This study provides a single basis for comparison and insights into circumstances where one technique will out-perform another. While [15] compares static mapping heuristics, eight dynamic on-line heuristics for mapping a class of independent tasks are compared in [4] [40] [39]. In contrast to static, off-line mappers, which assume a knowledge of what tasks are to be planned for execution during the next day (or other time interval), dynamic, on-line mappers handle tasks as they arrive (without such prior knowledge). Three of the dynamic heuristics compared have been proposed as part of this research. The comparisons show that the selection of a dynamic mapping heuristic in a particular HC environment depends on the arrival rate of tasks and the optimization requirements.

A taxonomy for classifying different matching and scheduling methodologies is given in [16]. This taxonomy may be used to help classify and distinguish the different algorithms available with the MSHN Scheduling Advisor. A framework for simulating different HC environments to allow testing of relative performance of different mapping heuristics under different circumstances is presented in [5]. The paper characterizes an HC environment by using the expected execution times of the tasks that arrive in the system and maps them onto the different machines present in the system.

We contributed a chapter [53] to the upcoming book entitled *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*. This chapter summarizes our research that utilized genetic algorithms, including (1) the static use of a genetic algorithm for mapping communicating subtasks, (2) the use of a genetic algorithm to find "off-line mappings to use on-line" in certain environments, and (3) the comparison of eleven different static mapping heuristics (one of which was a genetic algorithm).

We present a summary of our genetic algorithm research for static mappings, our "on-line use of off-line mappings" for the dynamic use of precomputed mappings in certain environments, and the initial stages of our dynamic remapping study in an invited paper [51]. A summary of our genetic algorithm research for static mappings and our dynamic remapping studies is given in the invited keynote paper [43]. An invited journal paper [50] gives a review of some of our earlier mixed-machine HC research. This includes (1) characterization of techniques for mapping tasks on HC systems, (2) the MSHN architecture, and (3) comparisons of various static and dynamic mapping heuristics.

3.3 Performance of Mapping Algorithms

In [6] we studied the performance of four mapping algorithms. The four algorithms include two naive ones: Opportunistic Load Balancing (OLB), and Limited Best Assignment (LBA), and two intelligent greedy algorithms. All of these algorithms, except OLB, use expected run-times to assign jobs to machines. As expected run-times are rarely deterministic in modern networked and server based systems, we first use experimentation and an algorithmic approach [19] to determine some plausible run-time distributions. Using these distributions, we next execute simulations to determine how the mapping algorithms perform. Performance comparisons show that the greedy algorithms produce schedules that, when executed, perform better than naive algorithms, even though the exact run-times are not available to the schedulers. We conclude that the use of intelligent mapping algorithms is beneficial, even when the expected time for completion of a job is not deterministic.

We also performed event simulation experiments to investigate the cost tradeoffs of scheduling jobs in "groups" versus scheduling each job as it arrives [17]. Our results show that if the utilization factor for the system is near 1.0 (viz, when the mean arrival rate is comparable to the total mean service rate of the processors), job grouping is more efficient than per-job scheduling.

4 Resource Modeling and Monitoring

The heart of the Scheduling Advisor component of MSHN is a "model" of the network resources and tasks for which it is responsible. MSHN uses this model to make mapping decisions. The model's data is maintained in the Resource Requirements Database and the Resource Status Server. The effectiveness of MSHN's resource management services depends on how well it can model and monitor its resources and tasks. This section introduces several MSHN Project papers regarding our research into effective techniques for resource modeling and monitoring.

In [34] we determine, through simulation, that providing a more accurate estimate of the network load could permit users of adaptive applications to obtain better performance. We studied the accuracy with which resource loading information, particularly network loading information, must be known in order for applications to successfully, and with agility, adapt [33]. We determine that under many normal conditions, fairly inaccurate estimates of currently available bandwidth suffice. However, when the system is heavily loaded, some strategies can perform much better with very accurate load estimates. The accuracy with which the available bandwidth must be known varies not only with inter-arrival rate, but also with the adaptation strategy used and the percentage of adaptive applications in the system.

In [48] we describe the design, implementation, and results of the first MSHN Client Library prototype. This research develops the mechanism and policy for the Client Library's resource monitoring role and carefully documents how applications can be easily linked with the Client Library. Additionally, we describe a policy for passively gathering network performance characteristics, i.e., latency and throughput, to minimize overhead added to the run-time of test programs.

In [47] we focus on the problem of monitoring the end-to-end performance of adaptive MSHN applications. Based upon a survey of available monitoring tools and analytical experiments, we conclude that the optimal monitoring mechanism: (1) should be passive; (2) should not require domain-specific knowledge of an application; (3) should minimize sources of error; and (4) should have few limitations. No single tool or application component surveyed has all of these characteristics. We describe a new tool whose mechanisms have all of the desired characteristics, and how we implemented it, in detail.

System models that are too detailed incur unnecessary overhead when values corresponding to the detail are being obtained; they are subject to higher variances; and the benefit of computing schedules using them may be outweighed by the time required to compute those schedules. In [18] we propose a model that balances the level of detail, and therefore the quality of their predictions of resource usage, against the cost of computing schedules. To assess the quality of the proposed model, an Application Emulator was designed, built, and used. The results from running the Application Emulator demonstrated that the proposed model is able to predict the relative resource usage of an asynchronous application that has substantially more computation requirements than communication requirements. We refined this model in [49] to correctly estimate the relative execution times of certain communication-intensive, and compute-intensive, asynchronous applications.

As part of our communication scheduling framework we developed an analytical communication model to compute the time for node-to-node communication events [8]. The model represents the network performance between processor pairs using two parameters: start-up cost and data transmission rate. The analytical communication model is represented in a timing diagram, which is input to the scheduling algorithm.

We investigated the capabilities of currently available communication resource status monitoring tools for the purpose of identifying those tools that, with low overhead, can provide accurate, end-to-end communication status information in a Windows NT environment [31]. The techniques used by the various tools are described and the methods for determining the accuracy of these tools are specified.

In [45] we investigated methods of transparently intercept operating system calls made by a robust C4I modeling application, the Extended Air Defense Simulation (EADSIM), to measure the resources required by that application. MSHN utilizes this type of information to determine which version of an application to execute while meeting operational deadlines. We provide the first such data gathered on a complex, contemporary, C4I/air defense model currently in use throughout the DoD, and provide conclusions regarding the trade-offs of computing resources and confidence in simulation outcomes.

5 Distributed Communications

The heterogeneous computing nodes in a metacomputing system are interconnected by several types of networks such as Ethernet, ATM, and FDDI, among others. Many of the metacomputing applications involve frequent and large volumes of data transfer among the nodes. The overall application performance therefore depends largely on the system's communication performance. Network heterogeneity and dynamic run-time variations in network performance present significant challenges for efficient communication.

In the context of such a heterogeneous system, our research addressed the problem of efficient collective communication wherein a group of nodes communicate among one another. We introduced a uniform framework [7] for developing communication schedules for these collective communication patterns. Our framework consisted of analytical models of the heterogeneous network, abstract representations of the communication pattern, and scheduling algorithms. Schedules were adapted at run-time, based on network performance information obtained from a directory service. Our analytical models represented the communication performance between a pair of nodes as the sum of latency and bandwidth components. These components varied from one pair of nodes to another.

Based on this framework we have derived efficient communication schedules for total-exchange [8] [10] cyclic redistribution [9] [11], broadcast, and multicast [12] [13]. Our scheduling algorithms incorporated techniques from bi-partite graph matching, spanning tree algorithms, and shop scheduling theory. For the total-exchange problem, the open shop algorithm developed schedules which had a bounded completion time of at most twice the optimal. For this problem, our simulation results showed performance improvements of up to a factor of 5 over previous approaches. For the cyclic redistribution problem, we have implemented the open shop algorithm on a Cray T3E. Our results showed consistent performance improvements of up to 60 percent, compared with a baseline algorithm. Our scheduling techniques for the broadcast and multicast problems were based on spanning tree algorithms. Performance improvements of over a factor of 10 were achieved.

6 Performance Metrics

In a distributed heterogeneous computing environment, users' tasks are allocated resources to simultaneously satisfy, to varying degrees, the tasks' different, and possibly conflicting, quality of service (QoS) requirements. When the total demand placed on system resources by the tasks, for a given interval of time, exceeds the resources available, some tasks will receive degraded service or no service at all. One part of a measure to quantify the success of a resource management system (RMS) in such a distributed environment is the collective value of the tasks completed during an interval of time, as perceived by the user, application, or policy maker. The Flexible Integrated System Capability (FISC) ratio introduced in [32] is a measure for quantifying this collective value. The FISC ratio is a multi-dimensional measure, and may include priorities, versions of a task or data, deadlines, situational mode, security, application- and domain-specific QoS, and dependencies. In addition to being used for evaluating and comparing RMSs, the FISC ratio can be incorporated as part of the objective function in a system's scheduling heuristics.

7 Security

The MSHN security architecture [57] [56] [22] is based upon separation of services into four distinct partially ordered privilege domains, and provides security support for authentication, communications security, access control and accountability. It is designed to take advantage of operating system support for domains, where available, and uses emerging public key technology as an nearterm (interim) solution.

A method for articulating network security functional requirements, and for measuring their fulfillment, is presented in [27] [37]. Using this method, security in a quality of service framework (QoSS) is discussed in terms of variant security mechanisms and dynamic security policies. It is also shown how QoSS can be represented in a network scheduler benefit function. Fundamental QoSS concepts are discussed in [28].

In [29] we present an analysis of the layered and variable security services and requirements presented to a resource management system. We provide a network system model for analyzing how user and application choices and limits can affect the overall security provided by the RMS. We also present a method for fairly measuring the effectiveness of an RMS in performing security allocation and assignments with respect to security choices made by metacomputer users and applications

To knowledgeably assign computing and network resources to tasks, the resource management system (RMS) needs to know the resource-utilization costs associated with various network security services which it may assign to tasks. In [25] [26] we define a preliminary security service taxonomy defining the range of security services an RMS may need to manage; utilizing this taxonomy, we then provide a framework for defining the costs associated with network security services.

In [24] we address the problem of how users and administrators can understand and easily interact with a wide range of security services and mechanisms. We provide method for translation of a simplified user abstraction of security to detailed underlying mechanisms, such that users can be presented with a coherent user-level view of available security options.

We describe an approach for representing the level of resources consumed by jobs under the control of a resource management system [36], and it is shown how this measurement of resource usage can be combined with a notion of user preferences to reflect a restrictive resource-usage policy for network management.

8 Demonstration/Implementation

The MSHN Prototype consists of several inter-communicating components [23], the functions of which are described in the various MSHN documents. Development of the MSHN Scheduling Advisor component occurred at the Noemix site, and the other components were developed at NPS. Component integration was supported by both Noemix and NPS. In the Fall of 1999 all of the MSHN Prototype development and integration was transferred to the Noemix site. The following MSHN papers and theses describe various implementation issues regarding the prototype demonstration: use of CORBA, [20] [46] [21] real time support [44] system specification using UML [30] and Java threads [54].

9 Future Directions

9.1 Mapping Algorithms

Mapping research in progress builds on our past studies and results. We are developing techniques for mapping tasks to machines in heterogeneous environments where tasks have priorities, multiple versions, and deadlines. We are using a subset of FISC as the performance measure. We are designing two static mappers, selecting two that performed very well in our previous studies. After this is completed, possible future work would involve developing dynamic (on-line) mappers for such tasks and performance measure. We could also extend the performance measure to include security and application QoS attributes.

9.2 Security

The security architecture of the MSHN project may be applicable to other RMS architectures and to selected DoD applications. Additional work will be needed to understand how commercially available security architectures can be generalized for RMS support. The extension and development of the notion of Quality of Security Service is another area for further research. Theoretical work is needed to understand how QoSS can embrace survivability notions. The QoSS development of the MSHN project needs to be refined and applied to a variety of scheduling frameworks.

10 Acknowledgements

The participants in the MSHN effort are grateful for the support and guidance we have received from Dr. Gary Koob and his staff for the DARPA/ITO Quorum program. We wish to thank the many students, staff, and investigators at our respective institutions and the many other Quorum investigators who have provided us with insights and encouragement.

References

[1] Ammar Alhusaini, Viktor Prasanna, and C. S. Raghavendra. A Unified Resource Scheduling Framework for Heterogeneous Computing Environments. In *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW '99)*, pages 156–165, San Juan, PR, April 1999.

[2] Ammar Alhusaini, Viktor Prasanna, and C. S. Raghavendra. A Framework For Mapping With Resource Co-Allocation In Heterogeneous Computing Systems. In *Proceedings of the Ninth Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, To appear May, 2000.

[3] Ammar H. Alhusaini. *A Unified Mapping Framework For Heterogeneous Computing Systems*. PhD thesis, University of Southern California, Los Angeles, CA, In Progress 2000.

[4] Shoukat Ali. A comparative study of dynamic mapping heuristics for a class of independent tasks onto heterogeneous computing systems. M. S. thesis, Purdue University, West Lafayette, IN, August 1999.

[5] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Ali. Task Execution Time Modeling for Heterogeneous Computing Systems. In *Proceedings of the Ninth Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, To appear May, 2000.

[6] Robert Armstrong, Debra Hensgen, and Taylor Kidd. The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-Time Predictions. In *Proceedings of the Seventh Heterogeneous Computing Workshop (HCW '98)*, pages 79–87, Orlando, FL, March 1998.

[7] Prashanth B. Bhat. *Communication Scheduling Techniques For Distributed Heterogeneous Systems*. Ph. D. thesis, University Of Southern California, Los Angeles, CA, August 1999.

[8] Prashanth B. Bhat, Viktor K. Prasanna, and C. S. Raghavendra. Adaptive Communication Algorithms for Distributed Heterogeneous Systems. In *Proceedings of the 7th International Symposium On High Performance Distributed Computing (HPDC-7 '98)*, pages 310–321, Chicago, ILL, July 1998.

[9] Prashanth B. Bhat, Viktor K. Prasanna, and C. S. Raghavendra. Block-Cyclic Redistribution over Heterogeneous Networks. In *Proceedings of the 11th International Conference On Parallel And Distributed Computing Systems (PDCS '98)*, pages 242–249, Chicago, ILL, September 1998.

[10] Prashanth B. Bhat, Viktor K. Prasanna, and C. S. Raghavendra. Adaptive Communication Algorithms for Distributed Heterogeneous Systems. *Journal of Parallel and Distributed Computing*, 59(2):252–279, November 1999.

[11] Prashanth B. Bhat, Viktor K. Prasanna, and C. S. Raghavendra. Block-Cyclic Redistribution over Heterogeneous Networks. *Accepted for publication in Cluster Computing: The Journal of Networks, Software Tools and Applications*, To appear 2000.

[12] Prashanth B. Bhat, C. S. Raghavendra, and Viktor K. Prasanna. Efficient Collective Communication in Distributed Heterogeneous Systems. In *Proceedings of the 19th International Conference On Distributed Computing Systems*, pages 15–24, Austin, TX, May 1999.

[13] Prashanth B. Bhat, C. S. Raghavendra, and Viktor K. Prasanna. Efficient Collective Communication in Distributed Heterogeneous Systems. *Submitted to the Journal of Parallel and Distributed Computing*, To appear 2000.

[14] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladidlau L. Boloni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Technical Report TR.ECE 00-4, Purdue University, West Lafayette, IN, March 2000.

[15] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Boloni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A Comparison Study of Static Mapping Heuristics for a Class of Meta-Tasks on Heterogeneous Computing Systems. In *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW '99)*, pages 15–29, San Juan, Puerto Rico, April 1999.

[16] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Boloni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D Theys, and Bin Yao. A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 330–335, West Lafayette, IN, October 1998.

[17] James M. Breitinger. Optimal size of job pool for initiating a scheduling event. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1999.

[18] Paul F. Carff. When is a simple model adequate for use in scheduling in mshn. M. S. thesis, Naval Postgraduate School, Monterey, CA, March 1999.

[19] Thomas S. Cook. Dynamically determining distribution statistics in a distributed environment. M. S. thesis, Naval Postgraduate School, Monterey, CA, December 1999.

[20] Alpay Duman. The use and run-time overhead of corba in mshn project. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1998.

[21] Alpay Duman, Debra Hensgen, David St. John, and Taylor Kidd. Are CORBA Services Ready to Support Resource Management Middleware for Heterogeneous Computing. In *Proceedings of the Eighth Heterogeneous Computing Workshop(HCW '99)*, pages 83–96, San Juan, Puerto Rico, April 1999.

[22] John English. A management system for heterogeneous networks (mshn) security analysis. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1997.

[23] Debra A. Hensgen, Taylor Kidd, David St. John, Matthew C. Schnaidt, Howard Jay Siegel, Tracy D. Braun, Muthucumaru Maheswaran, Shoukat Ali, Jong-Kook Kim, Cynthia Irvine, Tim Levin, Richard F. Freund, Matt Kussow, Michael Godfrey, Alpay Duman, Paul Carff, Shirley Kidd, Viktor Prasanna, Prashanth Bhat, and Ammar Alhusaini. An Overview of MSHN: The Management System for Heterogeneous Networks. In *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW '99)*, pages 184–198, San Juan, Puerto Rico, April 1999.

[24] Cynthia Irvine and Tim Levin. A Note on Mapping User-Oriented Security Policies to Complex Mechanisms and Services. Technical Report NPS-CS-99-08, Naval Postgraduate School, Monterey, CA, June 1999.

[25] Cynthia Irvine and Tim Levin. Toward a Taxonomy and Costing Method for Security Services. In *Proceedings of the Computer Security Applications Conference*, pages 183–188, Phoenix, AZ, December 1999.

[26] Cynthia Irvine and Tim Levin. Toward a Taxonomy and Costing Method for Security Services. Technical Report NPS-CS-99-07, Naval Postgraduate School, Monterey, CA, June 1999.

[27] Cynthia Irvine and Tim Levin. An Introduction to Quality of Security Services. Technical Report NPS-CS-00-005, Naval Postgraduate School, Monterey, CA, April 2000.

[28] Cynthia Irvine and Tim Levin. The Effects of Security Choices and Limits in a Metacomputing Environment. Technical Report NPS-CS-00-004, Naval Postgraduate School, Monterey, CA, April 2000.

[29] Cynthia Irvine and Tim Levin. Toward Quality of Service in a Resource Management System Benefit Function. In *Proceedings of the Ninth Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, To appear May, 2000.

[30] David St. John, Taylor Kidd, Debra Hensgen, Mantak Shing, and Shirley Kidd. Experiences Using Semi-Formal Methods During Development of Distributed, Research-Oriented, System-Level Software. Technical Report NPS-CS-99-04, Naval Postgraduate School, Monterey, CA, May 1999.

[31] Ronald Jacobs Jr. Identifying accurate resource monitoring tools and techniques. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1999.

[32] Jong-Kook Kim, Debra A. Hensgen, Taylor Kidd, Howard Jay Siegel, David St. John, Cynthia Irvine, Tim Levin, N. Wayne Porter, Viktor K. Prasanna, and Richard F. Freund. A QoS Performance Measure Framework for Distributed Heterogeneous Networks. In *Proceedings of the Eighth Euromicro Workshop on Parallel and Distributed Processing*, pages 18–27, Rhodes, Greece, January 2000.

[33] John Kresho, Debra Hensgen, Taylor Kidd, and Geoffrey Xie. Determining the Accuracy Required in Resource Load Prediction to Successfully Support Application Agility. In *Proceedings of the Second IASTED European Parallel and Distributed Systems Conference (Euro-PDS 98)*, pages 224–254, Vienna, Austria, July 1998.

[34] John P. Kresho. Quality network load information improves performance of adaptive applications. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1997.

[35] Yu-Kwong Kwok, Anthony A. Maciejewski, Howard Jay Siegel, Arif Ghafoor, and Ishfaq Ahmand. Evaluation of a Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems. In

Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '99), pages 204–209, Fremantle, Australia, June 1999.

[36] Tim Levin and Cynthia Irvine. An Approach to Characterizing Resource Usage and User Preferences in Benefit Functions. Technical Report NPS-CS-99-005, Naval Postgraduate School, Monterey, CA, June 1999.

[37] Tim Levin and Cynthia Irvine. Quality of Security Service in a Resource Management System Benefit Function. Technical Report NPS-CS-00-02, Naval Postgraduate School, Monterey, CA, November 1999.

[38] Muthucumaru Maheswaran. *Software Issues On Mapping Applications Onto Heterogeneous Machines And The Performance Of Krylov Algorithms On Parallel Machines.* Ph. D. thesis, Purdue University, West Lafayette, IN, December 1998.

[39] Muthucumaru Maheswaran, Shaoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW '99)*, pages 30–44, San Juan, Puerto Rico, April 1999.

[40] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, November 1999.

[41] Muthucumaru Maheswaran, Tracy Braun, and Howard Jay Siegel. High Performance Mixed-Machine Heterogeneous Computing. In *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*, pages 3–9, Madrid, Spain, January 1998.

[42] Muthucumaru Maheswaran, Tracy D. Braun, and Howard Jay Siegel. *Heterogeneous Distributed Computing, Encyclopedia of Electrical and Electronics Engineering*, volume 8, pages 679–690. John Wiley and Sons, Inc., New York, NY, 1999.

[43] Muthucumaru Maheswaran and Howard Jay Siegel. A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems. In *Proceedings of the Seventh Heterogeneous Computing Workshop (HCW '98)*, pages 57–69, Orlando, FL, March 1998.

[44] Panagiotis Papadatos. Implementation of real-time mshn using ace and tao. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1999.

[45] N. Wayne Porter. Resource usage for adaptive c4i models in a heterogeneous computing environment. M. S. thesis, Naval Postgraduate School, Monterey, CA, June 1999.

[46] Matt Schnaidt, Alpay Duman, and Ted Lewis. A Comparison of C++ Sockets and Corba in a Distributed Matrix Multiply Application. Technical Report NPS-CS-99-001, Naval Postgraduate School, Monterey, CA, June 1998.

[47] Matt Schnaidt, Debra Hensgen, John Falby, Taylor Kidd, and David St. John. Passive, Domain-Independent, End-to-End Message Passing Performance Monitoring to Support Adaptive Applications in MSHN. In *Proceedings of the Eighth International Symposium in High Performance Distributed Computing*, pages 3–9, Redondo Beach, CA, August 1999.

[48] Matthew C. L. Schnaidt. Design, implementation, and testing of mshn's application resource monitoring library. M. S. thesis, Naval Postgraduate School, Monterey, CA, December 1998.

[49] Blanca A. Shaeffer. Refining a task-execution time prediction model for use in mshn. M. S. thesis, Naval Postgraduate School, Monterey, CA, March 2000.

[50] Howard Jay Siegel and Shoukat Ali. Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems. *Journal of Systems Architecture, Special Issue on Heterogeneous Distributed and Parallel Architectures: Hardware, Software and Design Tools*, To appear 2000.

[51] Howard Jay Siegel and Muthucumaru Maheswaran. Mapping Tasks onto Heterogeneous Computing Systems. In *Proceedings of the IX Simposio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho (IX Brazilian Symposium on Computer Architectures - High Performance Computing)*, pages 3–17, Sao Paulo, Brazil, October 1997.

[52] Min Tan and Howard Jay Siegel. A Stochastic Model for Heterogeneous Computing and Its Application in Data Relocation Scheme Development. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1088–1101, November 1998.

[53] Mitchell D. Theys, Tracy D. Braun, Yu-Kwong Kwok, Howard Jay Siegel, and Anthony A. Maciejewski. *Mapping of Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach, in Solutions to Parallel and Distributed Computing Problems: Lessons From Biological Sciences*. John Wiley and Sons, Inc., To appear 2000.

[54] Pedro Tsai. Re-targeting the graze performance debugging tool for java threads and analyzing the re-targeting to automatically parallelized (fortran) code. M. S. thesis, Naval Postgraduate School, Monterey, CA, March 2000.

[55] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski. Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, November 1997.

[56] Roger Wright, David J. Shifflett, and Cynthia E. Irvine. Security Architecture for a Virtual Heterogeneous Machine. In *Proceedings of the Fourteenth Computer Security Applications Conference*, pages 167–177, Phoenix, AZ, December 1998.

[57] Roger E. Wright. Integrity architecture and security services demonstration for management system for heterogeneous networks. M. S. thesis, Naval Postgraduate School, Monterey, CA, September 1998.

A Unified Resource Scheduling Framework for Heterogeneous Computing Environments

Ammar H. Alhusaini * and Viktor K. Prasanna*
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
Ph: (213) 740-4483
{ammar + prasanna}@usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
Ph: (310) 336-1686
raghu@aero.org

Abstract

A major challenge in Metacomputing Systems (Computational Grids) is to effectively use their shared resources, such as compute cycles, memory, communication network, and data repositories, to optimize desired global objectives. In this paper we develop a unified framework for resource scheduling in metacomputing systems where tasks with various requirements are submitted from participant sites. Our goal is to minimize the overall execution time of a collection of application tasks. In our model, each application task is represented by a Directed Acyclic Graph (DAG). A task consists of several subtasks and the resource requirements are specified at subtask level. Our framework is general and it accommodates emerging notions of Quality of Service (QoS) and advance resource reservations. In this paper, we present several scheduling algorithms which consider compute resources and data repositories that have advance reservations. As shown by our simulation results, it is advantageous to schedule the system resources in a unified manner rather than scheduling each type of resource separately. Our algorithms have at least 30% improvement over the separated approach with respect to completion time.

1. Introduction

With the improvements in communication capability among geographically distributed systems, it is attractive to use diverse set of resources to solve challenging applications. Such Heterogeneous Computing (HC) systems [12, 17] are called *metacomputing* systems [26] or *computational grids* [8]. Several research projects are underway, including for example, MSHN [22], Globus [13], and Legion [19], in which the users can select and employ resources at different domains in a seamless manner to execute their applications. In general, such metacomputing systems will have compute resources with different capabilities, display devices, and data repositories all interconnected by heterogeneous local and wide area networks. A variety of tools and services are being developed for users to submit and execute their applications on a metacomputing system.

A major challenge in using metacomputing systems is to effectively use the available resources. In a metacomputing environment, applications are submitted from various user sites and share system resources. These resources include compute resources, communication resources (network bandwidth), and data repositories (file servers). Programs executing in such an environment typically consist of one or more subtasks that communicate and cooperate to form a single application. Users submit jobs from their sites to a metacomputing system by sending their tasks along with Quality of Service (QoS) requirements.

Task scheduling in a distributed system is a classic problem (for a detailed classification see [5, 6]). Recently, there have been several works on scheduling tasks in metacomputing systems. Scheduling independent jobs (meta-tasks) has been considered in [2, 11, 14]. For application tasks represented by Directed Acyclic Graphs (DAGs), many dynamic scheduling algorithms have been devised. These include the Hybrid Remapper [20], the Generational algorithm [9], as well as others [15, 18]. Several static algorithms for scheduling DAGs in metacomputing systems are described in [16, 23, 24, 25, 27]. Most of the previous algorithms focus on compute cycles as the main resource. Also, previous DAGs scheduling algorithms assume that a subtask receives all its input data from its predecessor subtasks. Therefore, their scheduling decisions are based on machine performance for the subtasks and the cost of receiving input

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

data from predecessor subtasks only.

Many metacomputing applications need other resources, such as data repositories, in addition to compute resources. For example, in data-intensive computing [21] applications access high-volume data from distributed data repositories such as databases and archival storage systems. Most of the execution time of these applications is in data movement. These applications can be computationally demanding and communication intensive as well [21]. To achieve high performance for such applications, the scheduling decisions must be based on all the required resources. Assigning a task to the machine that gives its best execution time may result in poor performance due to the cost of retrieving the required input data from data repositories. In [4], the impact of accessing data servers on scheduling decisions has been considered in the context of developing an AppLes agent for the Digital Sky Survey Analysis (DSSA) application. The DSSA AppLes selects where to run a statistical analysis according to the amount of required data from data servers. However, the primary motivation was to optimize the performance of a particular application.

In this paper we develop a unified framework for resource scheduling in metacomputing systems. Our framework considers compute resources as well as other resources such as the communication network and data repositories. Also, it incorporates the emerging concept of *advance reservations* where system resources can be reserved in advance for specific time intervals. In our framework, application tasks with various requirements are submitted from participant sites. An application task consists of subtasks and is represented by a DAG. The resource requirements are specified at the subtask level. A subtask's input data can be data items from its predecessors and/or data sets from data repositories. A subtask is ready for execution if all its predecessors have completed, and it has received all the input data needed for its execution. In our framework, we allow for input data sets to be replicated, i.e., the data set can be accessed from one or more data repositories. Additionally, a task can be submitted with QoS requirements, such as needed compute cycles, memory, communication bandwidth, maximum completion time, priority, etc. In our framework, sources of input data and the execution times of the subtasks on various machines along with their availability are considered simultaneously to minimize the overall completion time.

Although our unified framework allows many factors to be taken into account in resource scheduling, in this paper, to illustrate our ideas, we present several heuristic algorithms for a resource scheduling problem where the compute resources and the data repositories have advance reservations. These resources are available to schedule subtasks only during certain time intervals as they are reserved (by other users) at other times. QoS requirements such as deadlines and priorities will be included in future algorithms. The objective of our resource scheduling algorithms is to minimize the overall completion time of all the submitted tasks.

Our research is a part of the MSHN project [22], which is a collaborative effort between DoD (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). MSHN (Management System for Heterogeneous Networks) is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. Processes may have different priorities, deadlines, and compute characteristics. The goal is to schedule shared compute and network resources among individual applications so that their QoS requirements are satisfied. Our scheduling algorithms, or their derivatives, may be included in the Scheduling Advisor component of MSHN.

This paper is organized as follows. In the next section we introduce our unified resource scheduling framework. In Section 3, we present several heuristic algorithms for solving a general resource scheduling problem which considers input requirements from data repositories and advance reservations for system resources. Simulation results are presented in Section 4 to demonstrate the performance of our algorithms. Finally, Section 5 gives some future research directions.

2. The Scheduling Framework

2.1. Application Model

In the metacomputing system we are considering, n application tasks, $\{T_1, \ldots, T_n\}$, compete for computational as well as other resources (such as communication network and data repositories). Each application task consists of a set of communicating subtasks. The data dependencies among the subtasks are assumed to be known and are represented by a Directed Acyclic Graph (DAG), $G = (V, E)$. The set of subtasks of the application to be executed is represented by V=$\{v_1, v_2, \ldots, v_k\}$ where $v_k \geq 1$, and E represents the data dependencies and communication between subtasks. e_{ij} indicates communication from subtask v_i to v_j, and $|e_{ij}|$ represents the amount of data to be sent from v_i to v_j. Figure 1 shows an example with two application tasks. In this example, task 1 consists of three subtasks, and task 2 consists of nine subtasks.

In our framework, QoS requirements are specified for each task. These requirements include needed compute cycles, memory, communication bandwidth, maximum completion time, etc. In our model, a subtask's input data can be data items from its predecessors and/or data sets from data repositories. All of a subtask's input data (the data items and the data sets) must be retrieved before its execution. After

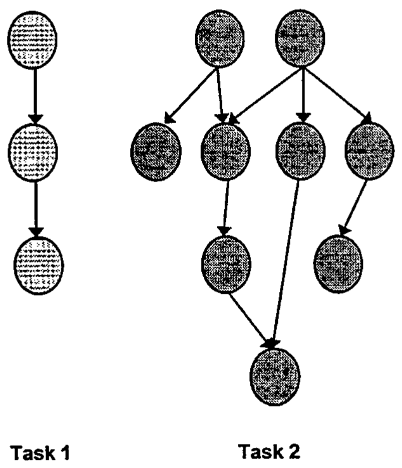


Figure 1. Example of application tasks

a subtask's completion, the generated output data may be forwarded to successor subtasks and/or written back to data repositories.

In some applications, a subtask may contain sub-subtasks. For example, Adaptive Signal Processing (ASP) applications are typically composed of a sequence of computation stages (subtasks). Each stage consists of a number of identical sub-subtasks (i.e., FFT's, QR decompositions, etc.). Each stage repeatedly receives its input from the previous stage, performs computations, and sends its output to the next stage.

2.2. System Model

The metacomputing system consists of m heterogeneous machines, $M = \{m_1, m_2, \ldots, m_m\}$, and f file servers or data repositories, $S = \{s_1, s_2, \ldots, s_f\}$. We assume that an estimate of the execution time of subtask v_i on machine m_j is available at compile-time. These estimated execution times are given in matrix ECT. Thus, $ECT(i,j)$ gives the estimated computation time for subtask i on machine j. If subtask v_i cannot be executed on machine m_j, then $ECT(i,j)$ is set to infinity.

System resources may not be available over some time intervals due to advance reservations. Available time intervals for machine m_j are given by $MA[j]$. Available time intervals for data repository s_j are given by $SA[j]$. Matrices TR and L give the message transfer time per byte and the communication latency between machines respectively. Matrices $Data\_TR$ and $Data\_L$ specify the message transfer time per byte and the communication latency between the data repositories and the machines, respectively. $DataSet[i]$ gives the amount of input data sets needed from data repositories for subtask v_i. In systems with multiple copies of data sets, one or more data repository can provide the required data sets for that subtask.

2.3. Problem Statement

Our goal is to minimize the overall execution time for a collection of applications that compete for system resources. This strategy (i.e., optimizing the performance of a collection of tasks as opposed to that of a single application) has been taken by SmartNet [11] and MSHN [22]. On the other hand, the emphasis in other projects such as AppLes [3] is to optimize the performance of an individual application rather than to cooperate with other applications sharing the resources. Since multiple users share the resources, optimizing the performance of an individual application may dramatically affect the completion time of other applications.

We now formally state our resource scheduling problem.

Given:

- A Metacomputing system with m machines and f data repositories,
- Advance reserved times for system resources as given by MA and SA,
- n application tasks, $\{T_1, \ldots, T_n\}$,where each application is represented by a DAG,
- Communication latencies and transfer rates among the various resources in matrices TR, L, $Data\_TR$, and $Data\_L$,
- Subtasks execution times on various machines in matrix ETC, and
- Amount of input data sets needed from data repositories for each subtask v_i as given by $DataSet[i]$.

Find a schedule to

$$Minimize\ \{\max_{j=1}^{n} [Finish\ Time(T_j)]\ \},$$

where the schedule determines, for each subtask, the start time and the duration of all the resources needed to execute that subtask.

Subject to the following constraints:

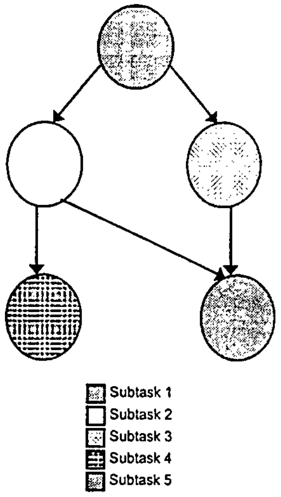


Figure 2. Application DAG for the example in Sec. 2.4

	m_1	m_2	m_3
V_1	5	4	8
V_2	20	5	3
V_3	6	10	4
V_4	10	4	2
V_5	∞	6	5

Table 1. Subtask execution times

- A subtask can execute only after all its predecessors have completed, all input data items have been received from its predecessors, and the input data sets have been retrieved from one of the data repositories,
- Preserve all advance resource reservations,
- Only one subtask can execute on any machine at any given time, and
- At most one subtask can access any data repository at any given time.

2.4. Separated Scheduling Vs. Unified Scheduling

Many scheduling methods exist in the literature for scheduling application DAGs on compute and network resources. They do not consider data repositories. With the inclusion of data repositories, one can obtain schedules for compute resources and data repositories independently and

	m_1	m_2	m_3
S_1	5	6	6
S_2	1	4	3
S_3	4	1.5	5

Table 2. Transfer costs (time units/data unit)

Subtask	Amount of the Input Data Set	Data Repository Choices
V_1	3 units	S_1 or S_2
V_2	10 units	S_2 or S_3
V_3	2 units	S_1 or S_3
V_4	1 unit	S_1 or S_2
V_5	5 units	S_3

Table 3. Input requirements for the subtasks

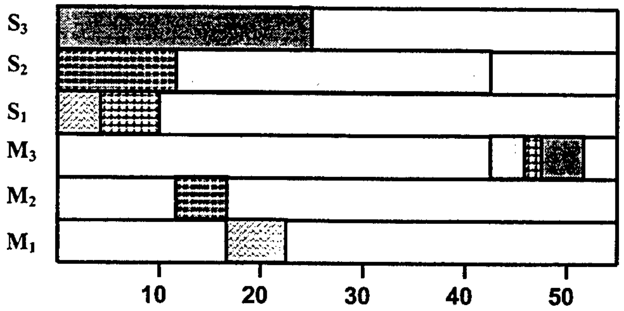


Figure 3. Separated scheduling (machines first)

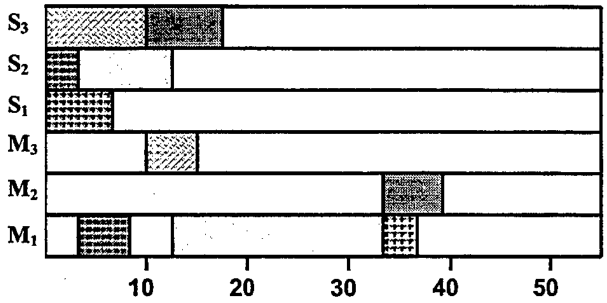


Figure 4. Separated scheduling (data repositories first)

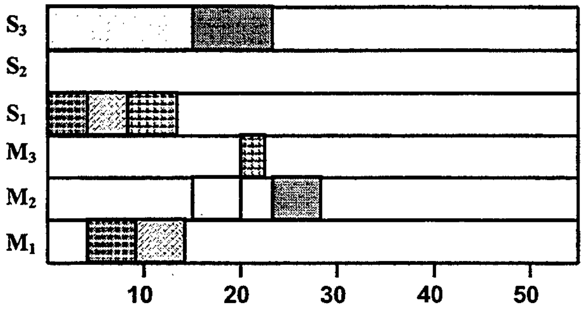


Figure 5. Unified scheduling

combine the schedules. In this section we show with a simple example, that this separated approach is not efficient with respect to completion time.

Figure 2 shows the DAG representation for an application task with 5 subtasks. In this example, we assume a fully connected system with 3 machines and 3 data repositories (file servers). The subtask execution times (in time units) are given in Table 1. Table 2 gives the the cost (in time units) for transferring one data unit from any data repository to any machine. We assume that each subtask needs an input data set, which can be retrieved from one or more data repositories as given in Table 3.

In this example, we are using a simple list scheduling algorithm called the Baseline Algorithm. This algorithm has been described in [20, 27]. The baseline algorithm is a fast static algorithm for mapping DAGs in HC environments. It partitions the subtasks in the DAG into blocks (levels) using an algorithm similar to the level partitioning algorithm which will be described in Section 3.1. Then all the subtasks are ordered such that the subtasks in block k come before the subtasks in block b, where $k < b$. The subtasks in the same block are sorted in descending order based on the number of descendents of each subtask (ties are broken arbitrarily). The subtasks are considered for mapping in this order. A subtask is mapped to the machine that gives the minimum completion time for that particular subtask. Since the original algorithm does not account for the data repositories, we implemented a modified version of the algorithm. In the modified version, the algorithm chooses a data repository that gives the best retrieving time of the input data set.

The schedule based on the separated approach, when scheduling the machines first, is shown in Figure 3. The completion time of this schedule is 52 time units. For this case, we map the application subtasks to the machines as they are the only resources in the system. Then for each subtask we choose the data repository that gives the best retrieving (delivery) time of the input data set to the previously mapped machine for this subtask in order to minimize its completion time. The completion time of the schedule based on the separated approach, when scheduling the data repositories first, is 39 time units as shown in Figure 4. For this case, we map the application subtasks to the data repositories as they are the only system resources. Then for each subtask we choose the machine that gives the best completion time for that subtask when using the previously mapped data repository to get the required data set for this subtask. Figure 5 shows the schedule based on the unified approach. The completion time of the unified scheduling is 28.5 time units. In the unified approach, we map each subtask to a machine and data repository at the same time in order to minimize its completion time.

The previous example shows clearly that the scheduling based on the separated approach is not efficient with respect to completion time. Further, with advance reservations, separated scheduling can lead to poor utilization of resources when one type of resource is not available while others are available.

3. Resource Scheduling Algorithms

In this section, we develop static (compile-time) heuristic algorithms for scheduling tasks in a metacomputing system where the compute resources and the data repositories have *advance reservations*. These resources are available to schedule subtasks only during certain time intervals as they are reserved (by other users) at other times. Although our framework incorporates the notion of QoS, the algorithms we present in this paper do not consider QoS. We are currently working on extending our scheduling algorithms to

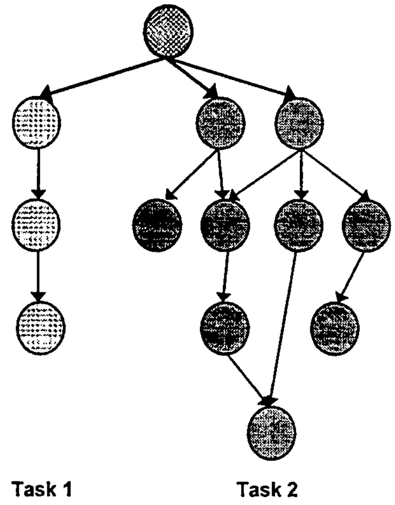


Figure 6. Combined DAG for the tasks in Fig. 1

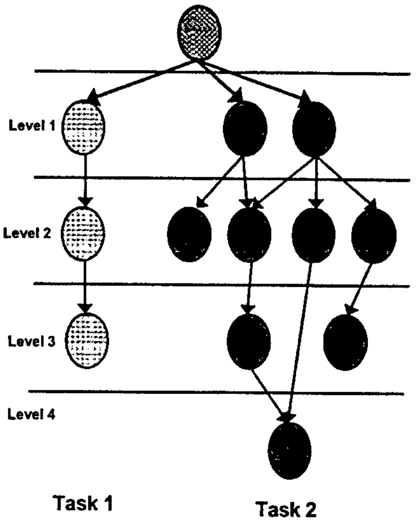


Figure 7. Level partitioning for the combined DAG in Fig. 6

consider QoS requirements such as deadlines, priorities, and security.

As in state-of-the-art systems, we assume a central scheduler with a given set of static application tasks to schedule. With static applications, the complete set of task to be scheduled is known a priori. Tasks from all sites are sent to the central scheduler to determine the schedule for each subtask so that the global objective is achieved. The information about the submitted tasks as well as status of various resources are communicated to the central scheduler. This centralized scheduler will then make appropriate decisions and can achieve better utilization of the resources.

Scheduling in metacomputing systems, even if we schedule based on compute resources only, is known to be NP-complete. One method is based on the well known list scheduling algorithm [1, 16, 23]. In list scheduling, all the subtasks of a DAG are placed in a list according to some priority assigned to each subtask. A subtask cannot be scheduled until all its predecessors have been scheduled. Ready subtasks are considered for scheduling in order of their priorities. In this section, we develop modified versions of list scheduling algorithm for our generalized task scheduling problem with advance resource reservations. Our heuristic algorithms that are based on the list scheduling are of two types – level by level scheduling and greedy approach. In the following, we briefly describe these two types of algorithms.

3.1. Level-By-Level Scheduling

In our framework, application tasks are represented by DAGs where a node is a subtask and the edges from predecessors represent control flow. Each subtask has computation cost, data items to be communicated from predecessor subtasks, and data sets from one or more repositories. A subtask is ready for execution if all its predecessors have completed, and it has received all the input data needed for its execution. To facilitate the discussion of our scheduling algorithms, a hypothetical node is created and linked, with zero communication time links, to the root nodes of all the submitted DAGs to obtain one combined DAG. This dummy node has zero computation time. Figure 6 shows the combined DAG for the two tasks in Figure 1. Now, minimizing the maximum time to complete this combined DAG achieves our global objective.

In level-by-level heuristic, we first partition the combined DAG into l levels of subtasks. Each level contains independent subtasks, i.e., there are no dependencies between the subtasks in the same level. Therefore, all the subtasks in a level can be executed in parallel once they are ready. Level 0 contains the dummy node. Level 1 contains all subtasks that do not have any incident edges originaly, i.e., subtasks without any predecessors in the original DAGs. All subtasks in level l have no successors. For each subtask v_j in level k, all of its predecessors are in levels 0 to $k-1$, and at least one of them in level k-1. Figure 7 shows the levels of the combined DAG in Fig. 6. The combined DAG in this example

```
Level-by-Level Scheduling Algorithm
begin
    Combine all submitted DAGs into one DAG.
    Do level partitioning for the combined DAG.
    For level := 1 to $l$ do
        Set $Ready$ to be the set of all subtasks at this level.
        While $Ready$ is not empty do
            Find $FINISH(v_i, m_{min}, s_{min})$ for all subtasks in $Ready$, where $m_{min}$ is
              the machine that gives the minimum completion time for subtask $v_i$
              if data repository $s_{min}$ has been used to get the input data set.
            Min-FINISH: Choose the subtask $v_k$ with the minimum completion time.
            Max-FINISH: Choose the subtask $v_k$ with the maximum completion time.
            Schedule subtask $v_k$ to machine $m_{min}$ and data repository $s_{min}$.
            Update $MA(m_{min})$ and $SA(s_{min})$.
            Remove $v_k$ from $Ready$.
        end While
    end For
end
```

Figure 8. Pseudo code for the level-by-level scheduling algorithms

has 4 levels.

The scheduler considers subtasks in each level at a time. Among the subtasks in a particular level i, the subtask with the minimum completion time will be scheduled first in the *Min-FINISH* algorithm and the subtask with the maximum completion time is scheduled first in the *Max-FINISH* algorithm. The advance reservations of compute resources and data repositories are handled by choosing the first-fit time interval to optimize the completion time of a subtask.

The idea behind the *Min-FINISH* algorithm, as in algorithm D in [14] and Min-min algorithm in SmartNet [11], is that at each step, we attempt to minimize the finish time of the last subtask in the ready set. On the other hand, the idea in the *Max-FINISH*, as in algorithm E in [14] and Max-min algorithm in SmartNet [11], is to minimize the worst case finishing time for critical subtasks by giving them the opportunity to be mapped to their best resources. The pseudo code for the level-by-level scheduling algorithms is shown in Figure 8.

3.2. Greedy Approach

Since the subtasks in a specific level i of the combined DAG belong to different independent tasks, by scheduling level by level we are creating dependency among various tasks. Further, the completion times of levels of different tasks can vary widely, and the level-by-level scheduling algorithms may not perform well. The idea in the greedy heuristics, *Min-FINISH-ALL* and *Max-FINISH-ALL*, is to consider subtasks in all the levels that are ready to execute in determining their schedule. This will advance execution of different tasks by different amounts and will attempt to achieve the global objective and provide good response times for short tasks at the same time. As before, we consider both the minimum finish time and the maximum finish time of all ready subtasks in determining the order of the subtasks to schedule.

The two greedy algorithms, *Min-FINISH-ALL* and *Max-FINISH-ALL* algorithm, are similar to *Min-FINISH* and *Max-FINISH* respectively. They only differ with respect to the *Ready* set. In the greedy algorithms, the *Ready* set may contain subtasks from several levels. Initially, the *Ready* set contains all subtasks at level 1 from all applications. After mapping a subtask, the algorithms check if any of its successors are ready to be considered for scheduling and add them to *Ready* set. A subtask cannot be considered for scheduling until all its predecessors have been scheduled.

4. Results and Discussion

For the generalized resource scheduling problem considered above, it is not clear which variation of the list scheduling will perform best. Our intuition is that scheduling subtasks by considering all resource types together will result in bounded suboptimal solutions. In order to evaluate the effectiveness of the scheduling algorithms discussed in Sections 3.1 and 3.2, we have developed a software simulator that calculates the completion time for each of them. The input parameters are given to the simulator as fixed values or as a range of values with a minimum and maximum value.

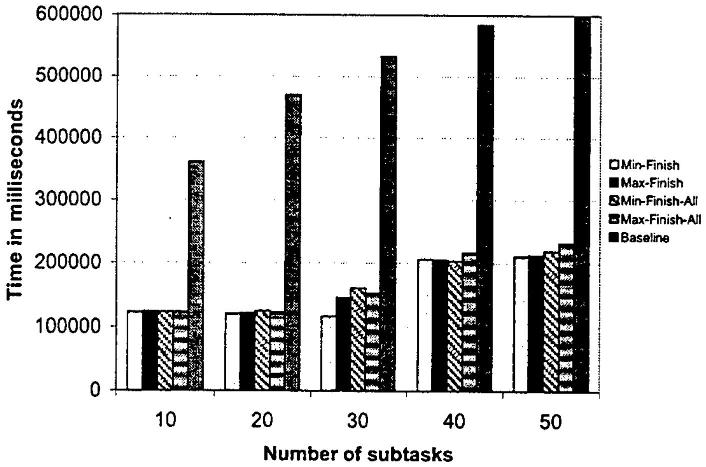


Figure 9. Simulation results for 20 machines and 6 data repositories with varying number of subtasks

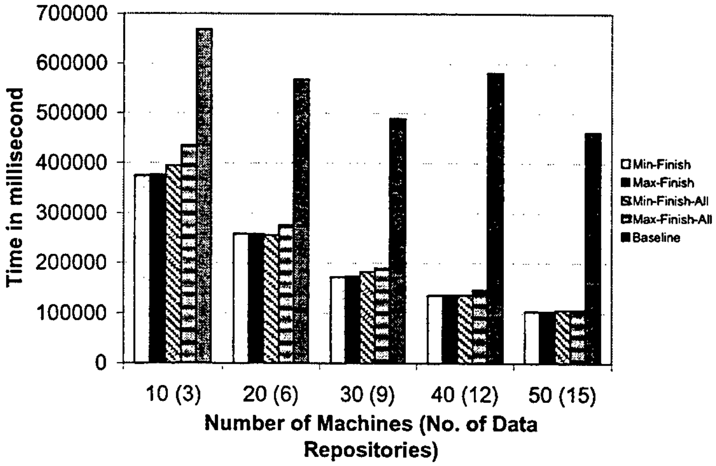


Figure 10. Simulation results for 50 subtasks with varying number of machines and data repositories

Subtask execution times, communication latencies, communication transfer rates, data items amounts, and data sets amounts, are specified to the simulator as range of values. The actual values of these parameters are choosen randomly by the simulator within the specified ranges. The fixed input parameters are the number of machines, the number of data repositories, the number of data items, and the total number of subtasks.

We assume that each task needs an input data set from the data repositories. This data set can be replicated and may be retreived from one or more data repositories. Each compute resource and data repository had several slots blocked at the beginning of the simulation to indicate advance reservations. We compare our scheduling algorithms with separated version of the baseline algorithm discussed in Section 2.4. The simulation results are shown in Figures 9 and 10. In Figure 9, the scheduling algorithms are compared for varying number of subtasks using 20 machines and 6 data repositories. Figure 10 shows a similar comparison for varying number of machines and data repositories with 50 subtasks. Our preliminary results show that all four of our heuristic algorithms seem to have similar performance with relatively uniform task costs. The simulation results clearly show that it is advantageous to schedule the system resources in a unified manner rather than scheduling each type of resource separately. Our scheduling algorithms have at least 30% improvment over the baseline algorithm which use the separated approach.

5. Future Work

This work represents, to the best of our knowledge, the first step towards a unified framework for resource scheduling with emerging constraints that are important in metacomputing. In this paper, we have considered one such requirement of advance reservations for compute resources and data repositories in this paper. We are investigating the question of how advance reservations impact task completion times. That is, in the scheduling, how soon we want to reserve a resource for a subtask to avoid waiting for another resource and/or blocking a different subtask. We are currently working on extending our scheduling algorithms to consider QoS requirements such as deadlines, priorities, and security. We are investigating the mapping of QoS specified at task level to subtasks in our framework.

In our future work we plan to develop scheduling algorithms for dynamic environments with the above mentioned resource requirements. In a dynamic environment, application tasks arrive in a real-time non-deterministic manner. System resources may be removed, or new resources may be added during run-time. Dynamic scheduling algorithms make use of real-time information and require feedback from the system.

References

[1] T. Adam, K. Chandy, and J. Dickson, "A comparison of list schedules for parallel processing systems," *Comm. of the ACM,* 17(12):685-690, Dec. 1974.

[2] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithm is independent of sizable variance in run-time predictions," *7th Heterogeneous Computing Workshop (HCW' 98),* pp. 79-87, March 1998.

[3] F. Berman and R. Wolski, "Scheduling from the perspective of the application," *5th IEEE International Symposium on High Performance Distributed Computing,* August 1996.

[4] F. Berman, "High-Performance schedulers," in *The Grid: blueprint for new computing infrastructure,* I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, San Francisco, CA, 1999, pp. 279-309.

[5] T. Braun et al., "A Taxonomy for describing matching and scheduling heuristics for mixed-machines heterogeneous computing systems," *Workshop on Advances in Parallel and Distributed Systems (APADS),* West Lafayette, IN, Oct. 1998.

[6] T. Casavant and J. Kuhl, " A Taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. on Software Engineering,* 14(2):141-154, Feb. 1988.

[7] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. on Software Engineering,* SE-15(11):1427-1436, Nov. 1989.

[8] I. Foster and C. Kesselman, ed., *The Grid: blueprint for new computing infrastructure,* Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[9] R. Freund, B. Carter, D. Watson, E. Keith, and F. Mirabile, "Generational scheduling for heterogeneous computing systems," *Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '96),* pp. 769-778, Aug. 1996.

[10] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous computing environments with SmarNet," *7th Heterogeneous Computing Workshop (HCW '98),* pp. 184-199, March 1998.

[11] R. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: a scheduling framework for heterogeneous computing," *The International Symposium on Parallel Architectures, Algorithms, and Networks,* Beijing, China, June 1996.

[12] R. Freund and H. J. Siegel, "Heterogeneous processing" *IEEE Computer,* 26(6):13-17, June 1993.

[13] Globus Web Page. *http://www.globus.org.*

[14] O. Ibarra and C. Kim, "Heuristic algorithms for scheduling independent tasks on non identical processors." *Journal of The ACM,* 24(2):280-289, April 1977.

[15] M. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment," *7th Heterogeneous Computing Workshop (HCW' 98),* pp. 70-78, March 1998.

[16] M. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW' 95),* pp. 93-100, Apr. 1995.

[17] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, " Heterogeneous computing: challenges and opportunities," *IEEE Computer,* 26(6):18-27, June 1993.

[18] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th Heterogeneous Computing Workshop (HCW' 95),* pp. 30-34, Apr. 1995.

[19] Legion Web Page. *http://legion.virginia.edu.*

[20] M. Maheswaran and H. J. Siegel, "A Dynamic matching and scheduling algorithm for heterogeneous computing systems," *7th Heterogeneous Computing Workshop (HCW '98),* pp. 57-69,March 1998.

[21] R. Moore, C. Baru, R. Marciano, A. Rajasekar, and M. Wan, "Data-intensive computing," in *The Grid: blueprint for new computing infrastructure,* I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, San Francisco, CA, 1999, pp. 105-129.

[22] MSHN Web Page. *http://www.mshn.org.*

[23] B. Shirazi, M. Wang, and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," *Journal of Parallel and Distributed Computing,* 10:222-232, 1990.

[24] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environment," *5th Heterogeneous Computing Workshop (HCW' 96),* pp. 98-117, Apr. 1996.

[25] G. C. Sih and E. A. Lee, "A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans.on Parallel and Distributed Systems,* 4(2):175-187, Feb. 1993.

[26] L. Smarr and C. E. Catlett, "Metacomputing," *Communications of the ACM,* 35(6):45-52, June 1994.

[27] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *5Journal of Parallel and Distributed Computing,* 47(1):8-22, Nov. 1997.

Biographies

Ammar Alhusaini is a Ph.D. candidate in the Department of Electrical Engineering - Systems at the University of Southern California, Los Angeles, California, USA. His main research interest is task scheduling in heterogeneous environments. Mr. Alhusaini received a B.S. degree in computer engineering from Kuwait University in 1993 and M.S. degree in computer engineering from the University of Southern California in 1996. Mr. Alhusaini is a member of IEEE, IEEE Computer Society, and ACM.

Viktor K. Prasanna (V.K. Prasanna Kumar) is a Professor in the Department of Electrical Engineering - Systems, University of Southern California, Los Angeles. He received his B.S. in Electronics Engineering from the Bangalore University and his M.S. from the School of Automation, Indian Institute of Science. He obtained his Ph.D. in Computer Science from Pennsylvania State University in 1983. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision. Dr. Prasanna has published extensively and consulted for industries in the above areas. He is widely known for his pioneering work in reconfigurable architectures and for his contributions in high performance computing for signal and image processing and image understanding. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He also serves on the editorial boards of the Journal of Parallel and Distributed Computing and IEEE Transactions on Computers. He is the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a Fellow of the IEEE.

Cauligi Raghavendra is a Senior Engineering Specialist in the Computer Science Research Department at the Aerospace Corporation. He received the Ph.D degree in Computer Science from University of California at Los Angeles in 1982. From September 1982 to December 1991 he was on the faculty of Electrical Engineering-Systems Department at University of Southern California, Los Angeles. From January 1992 to July 1997 he was the Boeing Centennial Chair Professor of Computer Engineering at the School of Electrical Engineering and Computer Science at the Washington State University in Pullman. He received the Presidential Young Investigator Award in 1985 and became an IEEE Fellow in 1997. He is a subject area editor for the Journal of Parallel and Distributed Computing, Editor-in-Chief for Special issues in a new journal called Cluster Computing, Baltzer Science Publishers, and is a program committee member for several networks related international conferences.

In the Proc. 9th IEEE Heterogeneous Computing Workshop (HCW 2000), Cancun, Mexico, May 1, 2000.

A Framework for Mapping with Resource Co-Allocation in Heterogeneous Computing Systems

Ammar H. Alhusaini * and Viktor K. Prasanna*
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
Ph: (213) 740-4483
{ammar + prasanna}@usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
Ph: (310) 336-1686
raghu@aero.org

Abstract

It is often the case in Heterogeneous Computing (HC) systems that an application requires multiple resources of different types to be allocated simultaneously. In general, this problem is the resource co-allocation problem. In this paper, we develop a general framework for mapping a collection of applications with resource co-allocation requirements. In our framework, application tasks have two types of constraints to be satisfied: precedence constraints and resource sharing constraints. We use a graph theoretic framework to capture these constraints. A Directed Acyclic Graph is used to represent precedence constraints of tasks within an application and a Compatibility Graph is used to represent resource sharing constraints among tasks of applications. Both these graphs are used to find maximal independent sets of tasks that can be executed concurrently.

The objective of the mapping is to minimize the overall schedule length for a given set of applications. We develop heuristic algorithms to solve the mapping problem with resource co-allocation constraints. We also provide a two-phase algorithm that can be used for run-time adaptation. We conducted extensive simulation experiments to evaluate the performance of our heuristic algorithms. Simulation results for our algorithms show a performance improvement of 10% to 30% over a baseline algorithm of list scheduling which considers only the precedence constraints and allocates tasks from the resulting order. This paper demonstrates the importance of considering the co-allocation requirements when mapping applications in heterogeneous computing environments including grid environments.

1. Introduction

In *Heterogeneous Computing* (HC) systems [8, 13, 20, 25, 26], a diverse set of resources are used in a coordinated and effective way to solve computationally challenging applications. Such systems are also called *metacomputing* systems [29] or *computational grids* [10]. In general, such HC systems have compute resources with different capabilities, input/output devices, data repositories, and other resources all interconnected by heterogeneous local and wide area networks. A major challenge in using HC systems is to effectively use all the available resources.

Mapping applications in HC system is a well researched problem in the literature. The mapping problem is defined as the problem of assigning application tasks to suitable resources (matching problem) and ordering task executions in time (scheduling problem) to optimize a specific objective function. Many algorithms exist for mapping applications in HC systems (for a detailed classification see [4]). For applications consisting of several tasks and represented by Directed Acyclic Graphs (DAGs), many static and dynamic mapping algorithms have been proposed. Dynamic algorithms include the Hybrid Remapper [23], the Generational algorithm [12], as well as others [1, 18, 21]. Several static algorithms for mapping application DAGs in HC systems are described in [19, 24, 27, 32]. Most of the previous algorithms focus on compute resources only.

In our earlier work [2], we developed a unified resource scheduling framework for HC systems which supports multiple resource requirements, advance reservation, and data replication. Each application was assumed to consist of several tasks and was represented by a DAG. A task's input data can be data items from its predecessors and/or data sets from data repositories. Input data sets can be accessed from one or more data repositories. Sources of input data and the execution times of the tasks on various machines along with

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

their availability were considered simultaneously to minimize the overall completion time. Although we considered multiple resource requirements in [2], tasks were not required to access different types of resources simultaneously.

In this paper, we consider the problem of mapping a set of applications in a HC system where application tasks require *concurrent access* to multiple resources of different types. In general, this problem is the *resource co-allocation* problem. For example, an interactive data analysis application may require simultaneous access to a storage system holding a copy of the data, a supercomputer for analysis, network elements for data transfer, and a display device for interaction [11]. For such applications, co-allocation of all required resources is necessary. A special case of this problem where a single application requires concurrent access to a set of resources in a computational grid has been considered in [5].

In this paper, we develop a general framework for mapping with resource co-allocation in HC systems. The framework defines the system and application models and formulates the co-allocation problem. Two graphs are used to represent applications: a Directed Acyclic Graph (DAG) and a *Compatibility Graph* (defined in Section 3.4). DAG representation is given initially and stay unchanged throughout the mapping process while the compatibility graph is updated during the mapping process. In classical mapping problems, only DAGs are used to represent the precedence constraints among tasks. In this paper, the co-allocation requirements add another type of constraint among the tasks: the resource sharing constraint which is captured in the compatibility graph. Tasks that share one or more resources cannot be executed concurrently due to the resource sharing constraints even if they have no precedence constraints among them. Known mapping algorithms for the classical DAG scheduling problem cannot be directly used for the above problem since they only consider the precedence constraints. In this paper, we develop heuristic algorithms that can be used with different allocation techniques to efficiently solve the co-allocation problem defined by our framework.

In our approach, multiple DAGs of different applications are combined into a single DAG. All tasks that have satisfied the precedence constraints are ready for allocation provided they have no resource sharing constraints. Using the compatibility graph, we will select tasks that can be executed concurrently. This is achieved by finding maximal independent sets in the compatibility graph.

Our research is part of the MSHN project [16], which is a collaborative effort between DoD (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). MSHN (Management System for Heterogeneous Networks) is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. Processes may have different priorities, deadlines, and compute characteristics. The goal is to schedule shared resources among individual applications so that their Quality of Service (QoS) requirements are satisfied. MSHN supports adaptive applications that can exist in several different versions. These versions may differ in the precision of computation or input data, and therefore have different values to a user. Unlike other HC and grid projects, MSHN seeks to determine how to meet QoS requirements of multiple application simultaneously.

The rest of this paper is organized as follows. In next section we give the definition of the co-allocation problem and summarize some related work. The problem framework is defined in Section 3. In Section 4, we give the outline of our approach to solve the co-allocation problem using our framework. Experimental results are given in Section 5. Finally, Section 6 gives the conclusions and future research directions.

2. The Co-Allocation Problem

The co-allocation problem can be defined as the problem of simultaneously allocating multiple resources of different types to applications in order to meet specific performance requirements. The need of co-allocation is a common characteristic of applications running in HC environments (as well as computational grids). For example, an application may require a data repository, a HPC platform, multiple display devices, and communication links all to be allocated simultaneously.

A version of resource co-allocation has been introduced in the high-performance distributed computing community by the Globus project [5]. The co-allocation problem is defined as the provision of allocation, configuration, and monitoring/control functions for the resource ensemble required by a single application [5]. The Globus tool-kit provides a flexible set of co-allocation mechanisms that can be used to construct application-specific co-allocation strategies. Only compute resources are considered in the Globus project at this time, to synchronize the start of complex applications at multiple sites.

The notion of co-allocation was also considered in the Legion project [22]. In the Legion project, an *Enactor* provides a mechanism to co-allocate compute and storage resources (hosts and vaults) to a single application. The co-allocation mechanism is based on advance resource reservation.

In [5] and [22], the focus is on implementation issues of the co-allocation process. Algorithms for efficient mapping with co-allocation requirements are not considered. Also, both the above projects focus on executing a single application. The problem becomes challenging when a number of applications share resources.

In this paper, we study the co-allocation problem in the context of mapping a set of applications where each application is represented by a DAG. We consider conflicts among tasks caused by precedence constraints as well as due to resource sharing. The objective is to minimize the overall schedule length for a set of applications. One of our main contributions in this paper is the formulation of the mapping problem in the presence of co-allocation requirements for multiple applications. To the best of our knowledge, this work is the first step towards a general framework for mapping applications with resource co-allocation in HC systems.

3. The Framework

3.1 System Model

We consider a heterogeneous computing system with m compute resources (machines), $M=\{m_1, m_2, \ldots, m_m\}$, and a set of r resources, $R=\{r_1, r_2, \ldots, r_r\}$. Compute resources can be HPC platforms, workstations, personal computers, etc. Resource $r_k \in R$ can be a data repository, an input/output device, etc. We assume that only one task can use any resource (compute and non-compute resource) at any given time. Resources are interconnected by heterogeneous communication links. Communication costs are given by two matrices: MM_comm and RM_comm, where MM_comm gives the communication cost for transferring a byte between machines and RM_comm gives the communication cost for transferring a byte between the resources in R and the machines.

We assume that an estimate of the computation time of a given task t_i on machine m_j is available at compile-time. These estimated computation times are given in an *Estimated Computation Time* (*ECT*) matrix. Thus, $ECT(t_i, m_j)$ gives the estimated computation time for task t_i on machine m_j. If task t_i cannot be executed on machine m_j, then $ECT(t_i, m_j)$ is set to infinity.

$MA(m_j)$ gives the earliest time when machine m_j is available and $RA(r_k)$ gives the earliest time when resource r_k is available. As the mapping proceeds, the earliest time when a resource (m_j or r_k) is available is calculated as the finish time of the last task assigned to this resource.

3.2. Application Model

In this HC system, a set of N applications, $A=\{A_1, \ldots, A_N\}$, compete for system resources. Each submitted application consists of several tasks and is modeled by a DAG, where the nodes represent computational requirements and the edges represent both precedence constraints and communication requirements. Figure 1 shows an example of two application DAGs. We assume that the whole set of applications to be mapped is known *apriori* (static applications). The problem is to execute these N applications as efficiently as possible. Our approach is to combine all submitted application DAGs into a single DAG, $G = (T, E)$, where T represents the set of tasks to be executed from all applications, $T=\{t_1, t_2, \ldots, t_n\}$, and E represents the data dependencies and communication between tasks. Edge e_{ij} indicates that there is communication from task t_i to t_j and its weight denotes the amount of communication. G is constructed by connecting the root nodes (tasks) of all applications to a hypothetical zero-cost entry node with zero-weight edges.

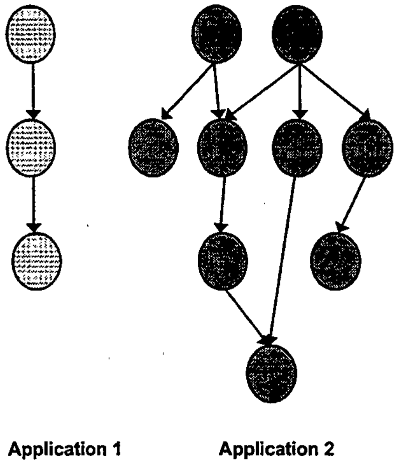


Figure 1. An example of two application DAGs

We assume that each task t_i needs *concurrent access* to a set of resources: one compute resource m_j and a number of additional resources as specified by the set $R(t_i)$, where $R(t_i) \subseteq R$. The amount of data to be transferred between t_i and r_k, where $r_k \in R(t_i)$, is given by $DATA(t_i, r_k)$. A task t_i cannot start execution until all its required resources are available to the task. All required resources will be allocated to the task during its execution. These resources will be available after the task completes its execution. We assume that all required resources are acquired at the same time (atomic transaction).

We say that task t_i and task t_j are *incompatible* if and only if $R(t_i) \cap R(t_j) \neq \phi$. Incompatible tasks cannot be executed concurrently even if they have no precedence constrains among them. Therefore, in our framework, tasks may be unable to run concurrently for either of the following reasons: (1) precedence constraints, or (2) resource sharing constraints.

The execution time of task t_i on machine m_j, $Exec(t_i, m_j)$, depends on the computation time of t_i on m_j and data transfer times between m_j and all resources which t_i needs to access during its execution. For example, for systems that assume computation and communication cannot be overlapped, $Exec(t_i, m_j)$ can be defined as

$$Exec(t_i, m_j) = ECT(t_i, m_j) + \sum_{r_k \in R(t_i)} (DATA(t_i, r_k) \times RM\_comm(r_k, m_j))$$

where the last term gives the total time to transfer any required data between machine m_j and every resource $r_k \in R(t_i)$. $Exec(t_i, m_j)$ can also be defined in different ways to consider the overlapping between computation and communication as well as other communication models.

The average execution time of task t_i is defined as

$$\overline{Exec(t_i)} = \sum_{j=1}^{m} Exec(t_i, m_j)/m$$

$ST(t_i, m_j)$ and $FT(t_i, m_j)$ are the earliest *start time* and the earliest *finish time* of task t_i on machine m_j, respectively if t_i were to be mapped on m_j. $ST(t_i, m_j)$ is defined as

$$ST(t_i, m_j) = \max\{MA(m_j), Data\_Pred(t_i, m_j)\}$$

where $Data\_Pred(t_i, m_j)$ is the time when task t_i receives all the needed data from all tasks in its predecessor set, $Pre(t_i)$, if t_i is mapped onto machine m_j. $FT(t_i, m_j)$ is defined as

$$FT(t_i, m_j) = ST(t_i, m_j) + Exec(t_i, m_j)$$

3.3. Mapping Objective

The objective function in our framework is to determine an assignment (matching) of tasks to compute resources and schedule their executions based on all resource requirements such that the overall schedule length (or *makespan*) of all submitted applications is minimized while satisfying all

1. Application-specified precedence constraints and

2. Implied resource sharing constraints.

Thus, we can define our objective function as

$$Minimize\ \{\max_{i=1}^{N} [Finish\ Time(A_i)]\ \},$$

where $Finish\ Time(A_i)$ is the completion time of application A_i. Note that the resource sharing constraint is a dynamic constraint - it depends on tasks ready to be allocated and their resource requirements.

Task	Resource Requirements
V_1	r_1, r_2
V_2	r_2, r_3
V_3	r_3, r_5
V_4	r_1, r_4
V_5	r_4, r_5, r_6
V_6	r_6

Table 1. An example showing 6 tasks and their resource requirements

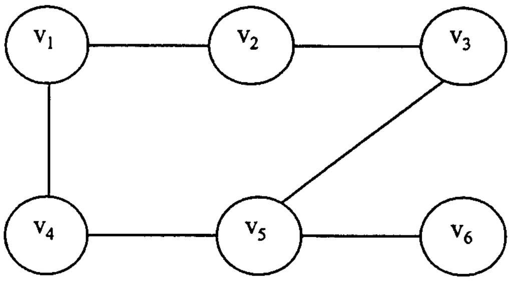


Figure 2. The compatibility graph for the tasks shown in Table 1

Task	Execution Time
V_1	5
V_2	6
V_3	2
V_4	4
V_5	1
V_6	3

Table 2. Execution times for the tasks in Figure 2

3.4. Compatibility Graph

To capture the implied resource sharing constraints among tasks that may belong to the same or different applications, we use the *compatibility graph*, $g = (V, E)$, where vertex v_i denotes task t_i and edge e_{ij} exists if and only if t_i and t_j are incompatible. Recall, task t_i and task t_j are incompatible if and only if $R(t_i) \cap R(t_j) \neq \phi$. An *independent set* [6] is a set of vertices of g such that no two vertices of the set are adjacent. An independent set is called a *maximal independent set* if there is no other independent set of g that contains it. A maximal independent set with the largest number of vertices among all maximal independent sets is called a *maximum independent* set [6]. The maximum independent set problem is NP-complete [15]. In our model, a maximal independent set of g represents a maximal set of tasks that can be executed concurrently if there is no precedence constraints among them.

As an example, consider a set of 6 independent tasks. Each task needs concurrent access to a set of resources as specified in Table 1. The compatibility graph g for this example is shown in Figure 2. The maximal independent sets of g are $\{V_1,V_5\}$, $\{V_2,V_5\}$, $\{V_1,V_3,V_6\}$, $\{V_2,V_4,V_6\}$, and $\{V_3,V_4,V_6\}$. The last three sets are maximum independent sets.

4. Our Solution

In classical DAG scheduling problem, application DAGs are partitioned onto levels such that each level contains independent tasks, i.e., there are no data dependencies among the tasks in the same level. Therefore, all tasks in the same level can be executed concurrently. In our framework, incompatible tasks in the same level cannot be executed concurrently due to resource sharing constraints. Therefore, mapping algorithms for the classical DAG scheduling problem (ex. [1, 30, 18, 23, 9, 31, 32]) cannot be directly used for our problem.

In this section, we develop a static co-allocation algorithm using the framework defined in Section 3. The algorithm can be used with different maximal independent set selection strategies and different allocation heuristics to solve the mapping problem with co-allocation requirements. Several strategies for selecting maximal independent sets and several allocation heuristics are given in this section. Also, we provide a two-phase algorithm that performs run-time adaptation.

4.1. The Co-Allocation Algorithm

Pseudo code for our co-allocation algorithm is shown in Figure 3. Given a set of applications and resource requirements of tasks, we first find tasks that have satisfied precedence constraints and then select maximal independent sets among these for allocation. The compatibility graph is used to find maximal independent sets. Since the maximum independent set problem is NP-complete [15], our approach for selecting a maximal independent set is based on first choosing a critical node v_c, and then finding a maximal independent set that contains v_c. Different strategies for selecting critical nodes are given in Section 4.2.

To ensure precedence constrains are satisfied, we combine all submitted applications into a single DAG, G, by using zero-weight edges to connect the root nodes (tasks) of all applications to a hypothetical zero-cost entry node. Then we partition the combined DAG onto l levels such that level 0 contains the entry node and level 1 contains all tasks that do not have any predecessors in the submitted DAGs. All tasks in level l have no successors. For each task t_i in level k, all of its predecessors are in levels 0 to $k-1$, and at least one of them in level k-1.

Let RDY be the set of tasks that have no precedence constraints among them and that are ready for allocation. A task is ready for allocation if for each predecessor all required resources have been allocated. Let W be the set of ready tasks that are waiting for allocation, and $ALLOCATED$ be the set of allocated tasks. After executing the algorithm, the list $SCHEDULE$ will give the resulting scheduling order of tasks and the variable $length$ will give the resulting schedule length.

In steps 1 and 2 of the algorithm, we combine all submitted applications into a single DAG, G, and partition G into l levels. Then the algorithm proceeds level-by-level as follows. For each level l of G, we construct the compatibility graph g for all tasks in this level (step 6). g is used to find maximal independent sets of tasks that can be executed concurrently. The first maximal independent set of tasks to be allocated is selected in steps 7-8 where a critical node v_c is chosen in step 7 and a maximal independent set that contains v_c is determined in step 8.

In step 10, all tasks in the selected maximal independent set are allocated to their required resources. For the allocation, we first find the scheduling order of the tasks. Several heuristics are given in Section 4.3. Then, we use this scheduling order to assign a compute resource m_j to each task t_i in order to minimize its finish time $FT(t_i, m_j)$. Availability times ($MA(m_j)$ and $RA(r_k)$) of all resources required by task t_i are updated based on $FT(t_i, m_j)$.

In steps 12-16, a new set of maximal independent tasks among all waiting tasks is selected to be allocated at the next allocation event. The next allocation event is calculated as the earliest finish time, $FT(t_i, m_j)$, among all allocated tasks. An allocated task v_x with the earliest finish time is identified in step 12 and then removed from the $ALLOCATED$ set. Initially (step 14), the set of candidate tasks that can be allocated next ,C, contains all waiting

Inputs: application DAGs, estimated computation and communication costs, and resource requirements of tasks
Outputs: the scheduling order of tasks ($SCHEDULE$) and the schedule length (makespan) ($lenght$) based on the given inputs

```
Begin
Combine all submitted application DAGs into a single DAG ($G$)
Do level partitioning of $G$    /* tasks in each level have no precedence constraints */
Let $SCHEDULE = \phi$ and $lenght = 0$
For level 1 to $l$ do
    Initialize $W$ to include all tasks in the current level and let $ALLOCATED = \phi$
    Construct the compatibility graph $g$ for all tasks in the current level
    Pick a critical node $v_c$ from $W$    /* several strategies can be used for critical node selection*/
    Find a maximal independent set of tasks $S$ from $W$ such that $v_c \in S$    /* $g$ is used to find the maximal independent set */
    While $W$ is not empty do
        Allocate all tasks in $S$ to their required resources by doing the following two steps:
            Find the scheduling order of the tasks and add them to $SCHEDULE$    /* different heuristics can be used */
            For each task $t_i$ in $S$ (in the scheduling order) do
                - Assign a compute resource $m_j$ to task $t_i$ in order to minimize its finish time $FT(t_i, m_j)$
                - Update $MA(m_j)$ and $RA(r_k), \forall r_k \in R(t_i)$, based on $FT(t_i, m_j)$
                - If ( $FT(t_i, m_j) > lenght$) then $lenght$=$FT(t_i, m_j)$
        Add all tasks in $S$ to $ALLOCATED$ and remove them from $W$
        Let $v_x$ be the allocated task with the lowest finish time
        Remove $v_x$ from $ALLOCATED$
        Let $C = W$, where $C$ is the set of candidate tasks that can be allocated next
        Remove all tasks from $C$ that are incompatible with any allocated task
        If ($C \neq \phi$)
            Pick a critical node $v_c$ from $C$ such that $v_c$ is adjacent to $v_x$ in $g$
            Find a maximal independent set of tasks $S$ from $C$ such that $v_c \in S$
    End (while)
End (for)
End
```

Figure 3. The co-allocation algorithm

tasks. The candidate set, C, is updated in step 15 by removing all tasks that are incompatible with any allocated task. Then g is used to find a maximal independent set of tasks from C. The algorithm repeats steps 10-16 until all tasks in this level have been allocated.

4.2. Maximal Independent Set Selection

Since the maximum independent set problem is NP-complete [15], we use a heuristic approach for selecting maximal independent sets. Our approach is based on first selecting a critical node v_c, and then finding a maximal independent set that contains v_c. Critical nodes need to be selected carefully.

The length of the schedule is influenced by the selection of maximal independent sets and by the order in which these sets are considered for scheduling. This is shown in the following example using the compatibility graph in Figure 2. For this example, for the sake of simplicity, we assume that all the resource requirements of tasks (compute and non-compute resources) are pre-specified. Therefore, the execution times of all tasks are known *apriori*. These times are shown in Table 2. Example schedules are given in Figures 4-6. These schedules have different schedule lengths. The optimal length of the schedule is 11 time units. This is achieved by schedules 2 and 3. In schedules 1 and 2, two different maximum independent sets were selected to be scheduled first. Schedule 1 has a length of 13 time units, while schedule 2 has the optimal length. This clearly shows the importance of the order in which the maximal independent sets are considered for scheduling. From schedule 3, we can also see that it is not always efficient to select a maximum independent set to be scheduled first. Schedule 3, which starts with a maximal independent set $\{V_2,V_5\}$ (not a maximum independent set), has the optimal length while schedule 1, which starts with a maximum set $\{V_3,V_4,V_6\}$, has a non-optimal length of 13 time units.

The idea behind our approach for selecting a maximal independent set S is to select a *critical* vertex v_c and add it to S which is initially empty. Then we attempt to enlarge S by traversing g. Different strategies can be used for selecting critical vertices. In the following we describe some of these strategies.

S1 Highest average execution time. In this strategy, we give priority for tasks that need more time for execution since they can be critical tasks. In HC systems, tasks have different execution times on different machines. Therefore, we use the average execution time $\overline{Exec(t_i)}$ as the selection criterion.

S2 Highest degree. The node out-degree in a DAG has been used in many list scheduling heuristics as a priority function. The out-degree of a node t_i gives the number of tasks that have precedence constraints with t_i. The idea is to advance the execution of tasks with high out-degree. Thus, many tasks can be ready for mapping once high out-degree tasks complete execution. In our framework, the out-degree of node t_i in the combined DAG G (which represents task t_i) does not reflect all dependencies between t_i and other tasks since G only captures the precedence constraints. Resource sharing constraints should also be considered. Therefore, we define the degree of task t_i as the sum of its out-degree in G and its degree in g. This number gives a better indication about the number of tasks that can be ready for mapping once t_i completes its execution, either because those tasks have a precedence or resource sharing dependencies with t_i.

S3 Critical path nodes. A Critical Path (CP) in a DAG is a path from an entry node to an exit node with the largest completion time. We use average execution times and average communication costs to find the critical path. In some situations, the average execution time or the degree of a task t_i cannot reflect how important for other tasks that t_i finishes execution as soon as possible. The successors of t_i may not be critical tasks and advancing their execution may not improve the schedule length. For these reasons, selecting critical path nodes from G as critical tasks can be a good strategy. In this paper, we implement two variations of this strategy:

S3.1 In this version, the task that is on the critical path is selected as a critical task. If there is no such task among the current set of candidate tasks, the task with the highest average execution time is selected as a critical task.

S3.2 This version is similar to S3.1 except for the case when there is no critical path node among the current set of candidate task. In this case, the task with the highest degree is selected as a critical task.

S4 Maximum weighted clique. In [3], a similar approach to the compatibility graph has been used for scheduling independent tasks. Each task in [3] requires simultaneous access to a set of pre-specified processors. All resource (processor) requirements and all execution times were assumed to be known *apriori*. It has been shown in [3] that the weight of the maximum weighted clique in the *constraint graph* (compatibility graph) is a lower bound on the optimum makespan, where the weight of each node is its execution time. In our previous example, notice that the weight of the maximum weighted clique (V_1 and V_2) is 11 and the optimal schedule length is 11. Also notice that any se-

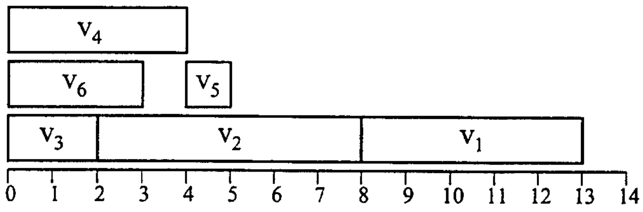


Figure 4. Schedule 1

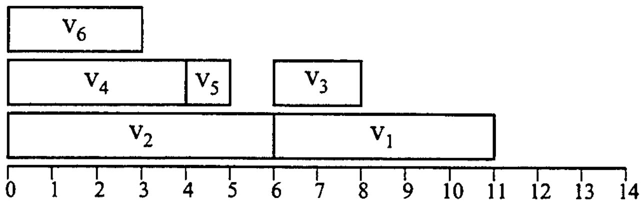


Figure 5. Schedule 2

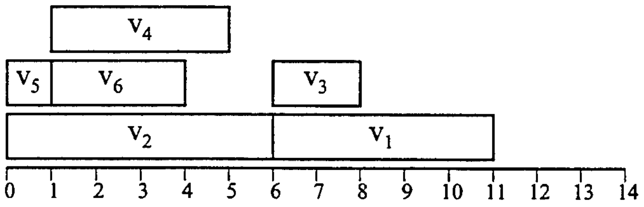


Figure 6. Schedule 3

lected maximal independent set should contain a task that belongs to the maximum weighted clique in order to achieve the optimal schedule length. Inspired by this observation, we can use nodes in the maximum weighted clique as candidates for selecting critical tasks. In our approach, we use the average execution times as the node weights. It is obvious that in our model, maximum weighted clique cannot guarantee the optimal solution but it could be a good heuristic for selecting maximal independent sets.

4.3. Allocation Heuristics

After selecting a maximal independent set of tasks, careful allocation of these tasks to compute resources (machines) is required to achieve our objective. Different heuristic can be used for allocating tasks of the selected maximal independent set S to compute resources. In the following we describe some of our allocation heuristics. The idea behind our heuristics is to advance the execution of tasks that may be critical in order to minimize the overall schedule length.

1. **Highest Average-Execution-Time First (HAETF).** In this heuristic, the average execution time is used as a priority function to place tasks in a list. All tasks are placed in a list in the order of non-increasing average execution times. Using this order, each task is allocated to the required resources such that its finish time is minimized.

2. **Maximum Finish-Time First (MAX).** For each task, we calculate the best finish time that can be achieved. Then we select the task with the maximum best finish time among all tasks. The task is allocated its required resources such that its finish time is minimized. We repeat until all tasks are allocated.

3. **Minimum Finish-Time First (MIN).** This heuristic is similar to the Maximum Finish-Time First (MAX) heuristic except that we select the task with the minimum finish time instead of selecting the task with the maximum finish time.

4. **Highest Degree First (HDF).** In this heuristic, all tasks are placed in a list in descending order according to their degrees (ties are broken arbitrarily). Then, tasks are allocated one-by-one to the required resources such that the finish time for each task is minimized.

4.4. Two-Phase Algorithm

We propose a two-phase algorithm for run-time adaptation using our static co-allocation algorithm. The two-phase algorithm can be used for the problem of mapping with resource co-allocation as defined in our framework as follows.

Phase 1: Compile-time mapping. At this phase, the co-allocation algorithm described in Section 4.1 is used to obtain an ordered list of tasks. The order of tasks in the list is based on their scheduling order as produced by our co-allocation algorithm. The list is obtained by satisfying all precedence and resource sharing constraints with the objective of minimizing the overall schedule length. Estimated computation and communication times are used to calculate the schedule length.

Phase 2: Run-time Adaptation. Run-time adaptation can be useful for the cases when actual execution times differ from the estimated execution times. One way to consider this is to scan through the ordered list obtained in phase 1 once a task completes its execution in order to find all tasks that can be executed at this time and make local reordering. The scanning can be done through a window of tasks with specific size k, where $k > 0$.

4.5. Implementation Issues

The focus of this paper is the mapping problem with resource co-allocation requirements in HC systems. The implementation details for the co-allocation process are outside the scope of this paper. A good discussion of implementation issue can be found in [5]. In the following for the sake of completeness, we briefly state our assumptions regarding the co-allocation implementation.

We assume that a task t_i cannot start execution until all its required resources are available. These resources will be acquired at the same time. Once a task t_i completes its execution, all its allocated resources will be released and will be available for other tasks. We assume that any allocation request for any resource will be granted as long as this resource is available. In this paper, we do not consider the cases of resource failures that can occur in the HC and Grid environments.

5. Performance Evaluation

A simulator was implemented to evaluate the performance of our co-allocation algorithms and the proposed selection strategies and allocation heuristics discussed in Section 4. In this section, we explain our simulation procedure and give experimental results.

5.1. Simulation Procedure

To define the HC system, numbers of machines and resources are given to the simulator as inputs. Communication costs among all resources are selected randomly from a uniform distribution with a mean equal to *ave_comm*. The communication costs are source and destination dependent.

The workload consists of randomly generated DAGs. Random DAGs are generated as follows: The number of tasks in the graph, *no_tasks*, maximum out-degree of a node, *max_outdegree*, average computation cost of a node, *ave_comp*, and average message size to be transferred among tasks, *ave_msg_size*, are given as inputs. First, the computation time of each task on every compute resource is randomly selected from a uniform distribution with a mean equal to *avg_comp*. Starting with the first task, the number of children (out-degree) is randomly selected between 1 and *max_outdegree*. Then, children are randomly selected for this task. The weight of each edge in the DAG is randomly selected from a uniform distribution with a mean equal to *ave_msg_size*. Resource requirements for each task are randomly selected from available resources. The amount of data to be transferred to/from each resource in the resource requirements set is randomly selected from a uniform distribution with a mean equal *ave_data_size*. The sizes of random DAGs range from 50 to 250 tasks with increments of 50.

5.2. Baseline Algorithm

Many mapping algorithms exist in the literature for mapping DAGs in HC systems. None of these algorithms consider the co-allocation problem we define in this paper. Therefore, we will use a simple list scheduling algorithm as a baseline algorithm to evaluate our co-allocation algorithm. The baseline algorithm is a fast static algorithm for mapping DAGs in HC environments. It partitions the tasks in the DAG into levels using an algorithm similar to the level partitioning algorithm described in Section 4.1. Then all the tasks are ordered such that the tasks in level k come before the tasks in level $k+1$. The tasks in the same level are sorted in descending order based on the average execution time of each task (ties are broken arbitrarily). The tasks are considered for mapping in this order. A task is mapped to the required resources such that its finish time is minimized.

The Baseline algorithm is similar to our algorithms in the sense that all algorithms proceed level-by-level. In the Baseline algorithm, the scheduling order of tasks at the same level is based on average execution times of tasks. On the other hand, the scheduling order in our algorithms is based on the selection of maximal independent sets. We construct the compatibility graph for each level and use it for selecting maximal independent sets. Different heuristics are used to order tasks within selected sets.

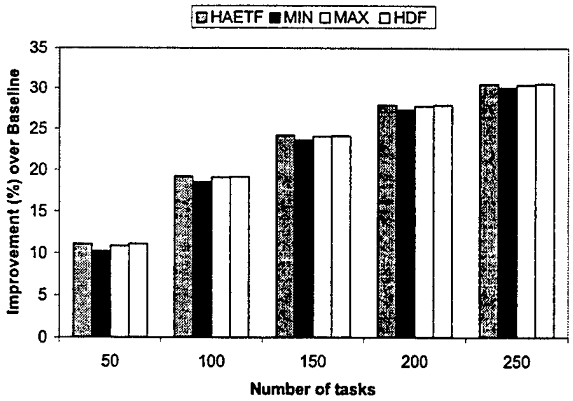


Figure 7. Performance of the allocation heuristics when using selection strategy $S1$

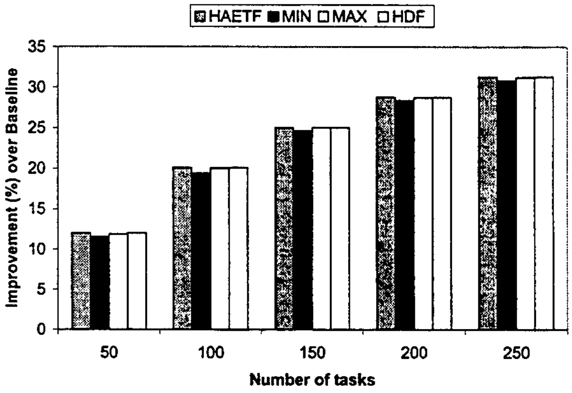


Figure 8. Performance of the allocation heuristics when using selection strategy $S2$

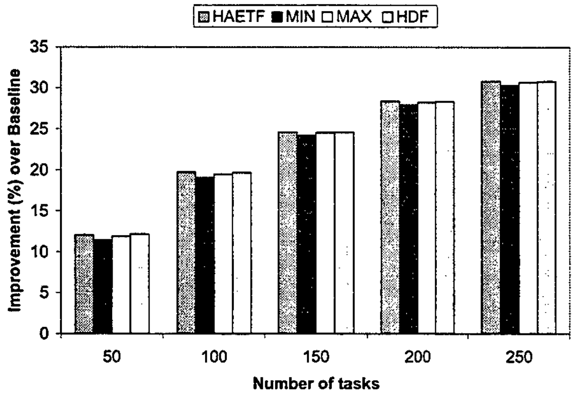


Figure 9. Performance of the allocation heuristics when using selection strategy $S3.1$

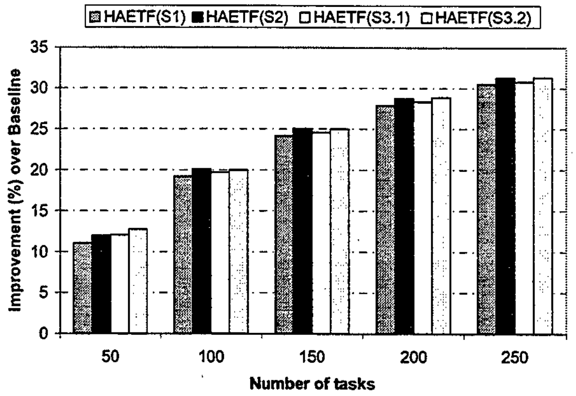


Figure 11. Performance of the selection strategies with $HAETF$ allocation heuristic

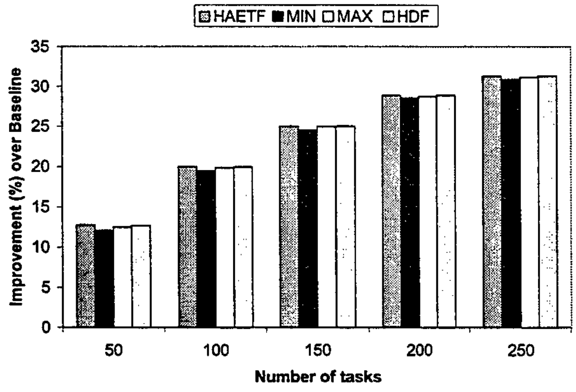


Figure 10. Performance of the allocation heuristics when using selection strategy $S3.2$

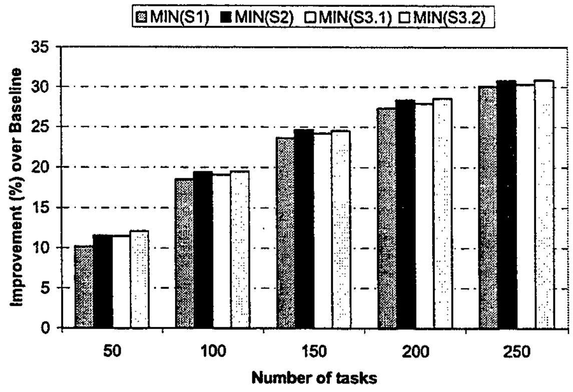


Figure 12. Performance of the selection strategies with MIN allocation heuristic

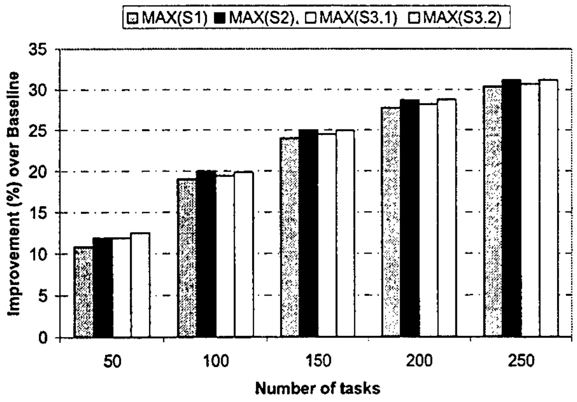


Figure 13. Performance of the selection strategies with MAX allocation heuristic

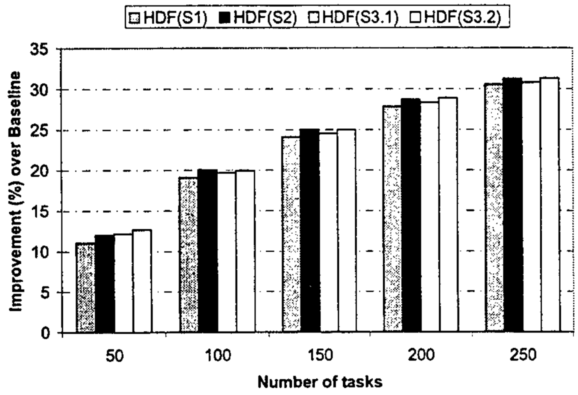


Figure 14. Performance of the selection strategies with HDF allocation heuristic

5.3. Experimental Results

Our experimental results are given in Figures 7- 14. The total number of tasks were varied from 50 to 250 with increments of 50. Each point in the figures is an average of 400 runs with different random DAGs. Random DAGs were generated with *max_outdegree*={2,3,4,5}, *ave_comp*=50, *ave_msg_size*=50K byte, and *ave_data_size*=300K byte.

Figures 7- 10 show the performance results of our allocation heuristics compared to the Baseline algorithm when using different maximal independent set selection strategies. The improvement of our heuristics over the Baseline increases as the total number of tasks increases. This shows the importance of considering co-allocation requirements in mapping algorithms. Generally, our allocation heuristics have relatively the same performance.

The performance results of maximal independent sets selection strategies when using different allocation heuristics are given in Figures 11- 14. As in the previous set of results, the improvement over Baseline algorithm increases as total number of tasks increases. Also, the selection strategies have relatively same performance.

In our simulation study, we found that the number of machines and the number of resources did not have a significant impact on the performance of allocation heuristics and selection strategies.

6. Conclusions and Future Work

This paper proposes a novel framework for the problem of mapping applications with resource co-allocation in HC systems. We formulated the co-allocation problem and developed several algorithms for solving this problem using a graph theoretic approach. Our simulation results show the importance of considering the co-allocation requirements during mapping decisions.

In solving our co-allocation problem, we need to find maximal independent sets among tasks competing for system resources. Although we considered many heuristics, they all seem to perform equally well indicating that a simple heuristic will suffice (even though one can create pathological examples for each heuristic).

In our future work, we plan to expand our framework to consider concurrent usage of multiple compute resources and *advance resource reservations*. With advance reservation, system resources can be reserved in advance for specific time intervals. Therefore, resource availability must be expressed as a list of available time slots and mapping algorithms should be insertion-based algorithms. To co-allocate a set of resources in this case, efficient algorithms are needed to find the best time slot when all resources are available for the required duration. In this paper, our algorithms are non insertion-based since the earliest available time for a resource r_i is the finish time of the last task assigned to r_i.

References

[1] I. Ahmad and Y. Kwok, "On parallelizing the multiprocessor scheduling problem," *IEEE Trans. on Parallel and Distributed Systems,* 10(4):414-432, Aprill 1999.

[2] A. Alhusaini, V. Prasanna, and C.S. Raghavendra, "A unified resource scheduling framework for heterogeneous computing environments,' ' *8th Heterogeneous Computing Workshop (HCW' 99),* pp. 156-165, April 12, 1999.

[3] L. Bianco, P. Dell'Olmo, and M. Grazia Sperenza, "Nonpreemptive scheduling of independent tasks with prespecified processor allocations," *Naval Research Logistics,* 41:959-971, 1994.

[4] T. Braun, H.J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao, "A Taxonomy for describing matching and scheduling heuristics for mixed-machines heterogeneous computing systems," *Workshop on Advances in Parallel and Distributed Systems (APADS),* West Lafayette, IN, Oct. 1998.

[5] K. Czajkowski, I. Foster, and C. Kesselman, "Resource co-allocation in computational grids," *7th IEEE Symposium on High Performance Distributed Computing,* pp. 219-228, 1999.

[6] N. Christofides, *Graph theory: An algorithmic approach,* Academic Press, 1975.

[7] Legion Web Page. *http://legion.virginia.edu*

[8] M. Eshagian (ed.), *Heterogeneous computing,* Norwood, MA: Artech House, 1996.

[9] M. Eshagian and Y.C. Wu, "Mapping heterogeneous task graphs onto heterogeneous system graphs," *6th Heterogeneous Computing Workshop (HCW' 97),* pp. 147-160, 1999.

[10] I. Foster and C. Kesselman, ed., *The Grid: blueprint for new computing infrastructure,* Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[11] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that support advance reservations a nd co-allocation," *Intl. Workshop on Quality of Service,* 1999.

[12] R. Freund, B. Carter, D. Watson, E. Keith, and F. Mirabile, "Generational scheduling for heterogeneous computing systems," *Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PD PTA '96),* pp. 769-778, Aug. 1996.

[13] R. Freund and H. J. Siegel, "Guest editors' introduction:Heterogeneous processing" *IEEE Computer,* 26(6):13-17, June 1993.

[14] Globus Web Page. *http://www.globus.org.*

[15] M. Gary and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness,* W.H. Freeman and Company, San Francisco, CA, 1979.

[16] D. Hensgen, T. Kidd, D. St.John, M. Schnaidt, H.J. Siegel, T. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: the Management System for Heterogeneous Networks," *8th Heterogeneous Computing Workshop (HCW' 99),* pp. 184-198, April 12, 1999.

[17] O. Ibarra and C. Kim, "Heuristic algorithms for scheduling independent tasks on non identical processors. " *Journal of The ACM,* 24(2):280-289, April 1977.

[18] M. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous en vironment," *7th Heterogeneous Computing Workshop (HCW' 98),* pp. 70-78, March 1998.

[19] M. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW' 95),* pp. 93-100, Apr. 1995.

[20] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: challenges and opportunities," *IEEE Computer,* 26(6):18-27, June 1993.

[21] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environm ent," *4th Heterogeneous Computing Workshop (HCW' 95),* pp. 30-34, Apr. 1995.

[22] Legion Web Page. *http://legion.virginia.edu.*

[23] M. Maheswaran and H. J. Siegel, "A Dynamic matching and scheduling algorithm for heterogeneous computing systems," *7th Heterogeneous Computing Workshop (HCW '98),* pp. 57-69, March 1998.

[24] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous e nvironment," *5th Heterogeneous Computing Workshop (HCW' 96),* pp. 98-117, Apr. 1996.

[25] H.J. Siegel, J. Antonio, R. Metzger, M. Tan, and Y. Li, " Heterogeneous computing," in *Parallel and distributed computing handbook,* A.Y. Zomaya (ed.), McGraw-Hill, New york, 1996, pp.725-761.

[26] H.J. Siegel, M. Mahesewaran, and T. Braun, "Heterogeneous distributed computing," in *Encyclopedia of electrical and electronics engineering,* J. Webster (ed.), John Wiley & Sons, New York, to appear.

[27] G. C. Sih and E. A. Lee, "A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans.on Parallel and Distributed Systems,* 4(2):175-187, Feb. 1993.

[28] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW' 96),* pp. 86-97 , April 15-16, 1996.

[29] L. Smarr and C. E. Catlett, "Metacomputing," *Communications of the ACM,* 35(6):45-52, June 1994.

[30] H. Topcuoglu, S. Hariri, and M. Wu, "Task scheduling algorithms for heterogeneous processors," *8th Heterogeneous Computing Workshop (HCW' 99),* pp. 3-14, April 12, 1999.

[31] R. Venkataramana and N. Ranganathan, "Multiple cost optimization for task assignment in heterogeneous computing systems using learning automata," *8th Heterogeneous Computing Workshop (HCW' 99),* pp. 137-145, April 12, 1999.

[32] L. Wang, H.J. Siegel, V. Roychowdhury, and A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genet ic-Algorithm-Based Approach," *Journal of Parallel and Distributed Computing,* 47(1):8-22, Nov. 1997.

Biographies

Ammar Alhusaini is a Ph.D. candidate in the Department of Electrical Engineering - Systems at the University of Southern California, Los Angeles, California, USA. His main research interest is task scheduling in heterogeneous environments. Mr. Alhusaini received a B.S. degree in computer engineering from Kuwait University in 1993 and M.S. degree in computer engineering from the University of Southern California in 1996. Mr. Alhusaini is a member of IEEE, IEEE Computer Society, and ACM.

Viktor K. Prasanna (V.K. Prasanna Kumar) is a Professor in the Department of Electrical Engineering - Systems, University of Southern California, Los Angeles. He received his B.S. in Electronics Engineering from the Bangalore University and his M.S. from the School of Automation, Indian Institute of Science. He obtained his Ph.D. in Computer Science from Pennsylvania State University in 1983. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision. Dr. Prasanna has published extensively and consulted for industries in the above areas. He is widely known for his pioneering work in reconfigurable architectures and for his contributions in high performance computing for signal and image processing and image understanding. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He also serves on the editorial boards of the Journal of Parallel and Distributed Computing and IEEE Transactions on Computers. He has the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a Fellow of the IEEE.

Cauligi Raghavendra is a Senior Engineering Specialist in the Computer Science Research Department at the Aerospace Corporation. He received the Ph.D degree in Computer Science from University of California at Los Angeles in 1982. From September 1982 to December 1991 he was on the faculty of Electrical Engineering-Systems Department at University of Southern California, Los Angeles. From January 1992 to July 1997 he was the Boeing Centennial Chair Professor of Computer Engineering at the School of Electrical Engineering and Computer Science at the Washington State University in Pullman. He received the Presidential Young Investigator Award in 1985 and became an IEEE Fellow in 1997. He is a subject area editor for the Journal of Parallel and Distributed Computing, Editor-in-Chief for Special issues in a new journal called Cluster Computing, Baltzer Science Publishers, and is a program committee member for several networks related international conferences.

Task Execution Time Modeling for Heterogeneous Computing Systems

Shoukat Ali[†], Howard Jay Siegel[†], Muthucumaru Maheswaran[‡],
Debra Hensgen[◇], and Sahra Ali[†]

[†]Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
{alis@ecn., hj@}purdue.edu

[‡]Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2 Canada
maheswar@cs.umanitoba.ca

[◇]OS Research and Evaluation
OpenTV
Mountain View, CA 94043 USA
dhensgen@opentv.com

Abstract

A distributed heterogeneous computing (HC) system consists of diversely capable machines harnessed together to execute a set of tasks that vary in their computational requirements. Heuristics are needed to map (match and schedule) tasks onto machines in an HC system so as to optimize some figure of merit. This paper characterizes a simulated HC environment by using the expected execution times of the tasks that arrive in the system onto the different machines present in the system. This information is arranged in an "expected time to compute" (ETC) matrix as a model of the given HC system, where the entry(i, j) is the expected execution time of task i on machine j. This model is needed to simulate different HC environments to allow testing of relative performance of different mapping heuristics under different circumstances. In particular, the ETC model is used to express the heterogeneity among the runtimes of the tasks to be executed, and among the machines in the HC system. An existing range-based technique to generate ETC matrices is described. A coefficient-of-variation based technique to generate ETC matrices is proposed, and compared with the range-based technique. The coefficient-of-variation-based ETC generation method provides a greater control over the spread of values (i.e., heterogeneity) in any given row or column of the ETC matrix than the range-based method.

1. Introduction

A distributed heterogeneous computing (<u>HC</u>) system consists of diversely capable machines harnessed together to execute a set of tasks that vary in their computational requirements. Heuristics are needed to <u>map</u> (match and schedule) tasks onto machines in an HC system so as to optimize some figure of merit. The heuristics that match a task to a machine can vary in the information they use. For example, the current candidate task can be assigned to the machine that becomes available soonest (even if the task may take a much longer time to execute on that machine than elsewhere). In another approach, the task may be assigned to the machine where it executes fastest (but ignores when that machine becomes available). Or the current candidate task may be assigned to the machine that completes the task soonest, i.e., the machine which minimizes the sum of task execution time and the machine ready time, where <u>machine ready time</u> for a particular machine is the time when that machine becomes available after having executed the tasks previously assigned to it (e.g., [13]).

The discussion above should reveal that more sophisticated (and possibly wiser) approaches to the mapping problem require estimates of the execution times of all tasks (that can be expected to arrive for service) on all the machines present in the HC suite to make better mapping decisions. One aspect of the research on HC mapping heuristics explores the behavior of the heuristics in different HC environments. The ability to test the relative performance of different mapping heuristics under different circumstances necessitates that there be a framework for generating simulated execution times of all the tasks in the HC system on all the machines in the HC system. Such a framework would, in turn, require a quantification of heterogeneity to express the variability among the runtimes of the tasks to be executed, and among the capabilities of the machines in the HC system. The goal of this paper is to present a methodology for synthesizing simulated HC environments with quantifiable levels of task and machine heterogeneity. This paper characterizes the HC environments so that it will be easier for the researchers to describe the workload and the machines used in their simulations using a common scale.

Given a set of heuristics and a characterization of HC

This research was supported by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and under the GSA subcontract number GS09K99BH0250. Some of the equipment used was donated by Intel.

environments, one can determine the best heuristic to use in a given environment for optimizing a given objective function. In addition to increasing one's understanding of the operation of different heuristics, this knowledge can help a working resource management system select which mapper to use for a given real HC environment.

This research is part of a DARPA/ITO Quorum Program project called MSHN (pronounced "mission") (Management System for Heterogeneous Networks) [7]. MSHN is a collaborative research effort that includes the Naval Postgraduate School, NOEMIX, Purdue, and University of Southern California. It builds on SmartNet, an implemented scheduling framework and system for managing resources in an HC environment developed at NRaD [5]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service. The methodology developed here for generating simulated HC environments may be used to design, analyze and evaluate heuristics for the Scheduling Advisor component of the MSHN prototype.

The rest of this paper is organized as follows. A model for describing an HC system is presented in Section 2. Based on that model, two techniques for simulating an HC environment are described in Section 3. Section 4 briefly discusses analyzing the task execution time information from real life HC scenarios. Some related work is outlined in the Section 5.

2. Modeling Heterogeneity

To better evaluate the behavior of mapping heuristics, a model of the execution times of the tasks on the machines is needed so that the parameters of this model can be changed to investigate the performance of the heuristics under different HC systems and under different types of tasks to be mapped. One such model consists of an expected time to compute (ETC) matrix, where the entry(i, j) is the expected execution time of task i on machine j. The ETC matrix can be stored on the same machine where the mapper is stored, and contains the estimates for the expected execution times of a task on all machines, for all the tasks that are expected to arrive for service over a given interval of time. (Although stored with the mapper, the ETC information may be derived from other components of a resource management system (e.g., [7])). In an ETC matrix, the elements along a row indicate the estimates of the expected execution times of a given task on different machines, and those along a column give the estimates of the expected execution times of different tasks on a given machine.

The exact actual task execution times on all machines may not be known for all tasks because, for example, they might be a function of input data. What is typically assumed in the HC literature is that estimates of the expected execution times of tasks on all machines are known (e.g., [6, 10, 12, 16]). These estimates could be built from task profiling and machine benchmarking, could be derived from the previous executions of a task on a machine, or could be provided by the user (e.g., [3, 6, 8, 14, 18]).

The ETC model presented here can be characterized by three parameters: machine heterogeneity, task heterogeneity, and consistency. The variation along a row is referred to as the machine heterogeneity; this is the degree to which the machine execution times vary for a given task [1]. A system's machine heterogeneity is based on a combination of the machine heterogeneities for all tasks (rows). A system comprised mainly of workstations of similar capabilities can be said to have "low" machine heterogeneity. A system consisting of diversely capable machines, e.g., a collection of SMP's, workstations, and supercomputers, may be said to have "high" machine heterogeneity.

Similarly, the variation along a column of an ETC matrix is referred to as the task heterogeneity; this is the degree to which the task execution times vary for a given machine [1]. A system's task heterogeneity is based on a combination of the task heterogeneities for all machines (columns). "High" task heterogeneity may occur when the computational needs of the tasks vary greatly, e.g., when both time-consuming simulations and fast compilations of small programs are performed. "Low" task heterogeneity may typically be seen in the jobs submitted by users solving problems of similar complexity (and hence have similar execution times on a given machine).

Based on the above idea, four categories were proposed for the ETC matrix in [1]: (a) high task heterogeneity and high machine heterogeneity, (b) high task heterogeneity and low machine heterogeneity, (c) low task heterogeneity and high machine heterogeneity, and (d) low task heterogeneity and low machine heterogeneity.

The ETC matrix can be further classified into two categories, consistent and inconsistent [1], which are orthogonal to the previous classifications. For a consistent ETC matrix, if a machine m_x has a lower execution time than a machine m_y for a task t_k, then the same is true for any task t_i. A consistent ETC matrix can be considered to represent an extreme case of low task heterogeneity and high machine heterogeneity. If machine heterogeneity is high enough, then the machines may be so much different from each other in their compute power that the differences in the computational requirements of the tasks (if low enough) will not matter in determining the relative order of execution times for a given task on the different machines (i.e., along a row). As a trivially extreme example, consider a system consisting of Intel Pentium III and Intel 286. The Pentium III will almost always run any given task from a certain set of tasks faster than the 286 provided the computational requirements of all tasks in the set are similar (i.e., low task heterogeneity), thereby giving rise to a consistent ETC matrix.

In inconsistent ETC matrices, the relationships among the task computational requirements and machine capabilities are such that no structure as that in the consistent case is enforced. Inconsistent ETC matrices occur in practice when: (1) there is a variety of different machine architectures in the HC suite (e.g., parallel machines, superscalars, workstations), and (2) there is a variety of different computational needs among the tasks (e.g., readily parallelizable tasks, difficult to parallelize tasks, tasks that are floating point intensive, simple text formatting tasks). Thus, the way in which a task's needs correspond to a machine's capabilities may differ for each possible pairing of tasks to machines.

A combination of these two cases, which may be more realistic in many environments, is the partially-consistent ETC matrix, which is an inconsistent matrix with a consistent sub-matrix [2, 13]. This sub-matrix can be composed of any subset of rows and any subset of columns. As an example, in a given partially-consistent ETC matrix, 50% of the tasks and 25% of the machines may define a consistent sub-matrix.

Even though no structure is enforced on an inconsistent ETC matrix, a given ETC matrix generated to be inconsistent may have the structure of a partially consistent ETC matrix. In this sense, partially-consistent ETC matrices are a special case of inconsistent ETC matrices. Similarly, consistent ETC matrices are special cases of inconsistent and partially-consistent ETC matrices.

It should be noted that this classification scheme is used for generating ETC matrices. Later in this paper, it will be shown how these three cases differ in generation process. If one is given an ETC matrix, and is asked to classify it among these three classes, it will be called a consistent ETC matrix only if it is fully consistent. It will be called inconsistent if it is not consistent.

Often an inconsistent ETC matrix will have some partial consistency in it. For example, a trivial case of partial-consistency always exists; for any two machines in the HC suite, *at least* 50% of the tasks will show consistent execution times.

3. Generating the ETC Matrices

3.1. Range Based ETC Matrix Generation

Any method for generating the ETC matrices will require that heterogeneity be defined mathematically. In the range-based ETC generation technique, the heterogeneity of a set of execution time values is quantified by the range of the execution times [2, 13]. The procedures given in this section for generating the ETC matrices produce inconsistent ETC matrices. It is shown later in this section how consistent and partially-consistent ETC matrices could be obtained from the inconsistent ETC matrices.

```
(1) for $i$ from 0 to $(t-1)$
(2)     $\tau[i] = U(1, R_{task})$
(3)     for $j$ from 0 to $(m-1)$
(4)         $e[i,j] = \tau[i] \times U(1, R_{mach})$
(5)     endfor
(6) endfor
```

Figure 1. The range-based method for generating ETC matrices.

Assume m is the total number of machines in the HC suite, and t is the total number of tasks expected to be serviced by the HC system over a given interval of time. Let $U(a, b)$ be a number sampled from a uniform distribution with a range from a to b. (Each invocation of $U(a, b)$ returns a new sample.) Let R_{task} and R_{mach} be numbers representing task heterogeneity and machine heterogeneity, respectively, such that higher values for R_{task} and R_{mach} represent higher heterogeneities. Then an ETC matrix $e[0..(t-1), 0..(m-1)]$, for a given task heterogeneity and a given machine heterogeneity, can be generated by the range-based method given in Figure 1, where $e[i,j]$ is the estimated expected execution time for the task i on the machine j.

As shown in Figure 1, each iteration of the outer **for** loop samples a uniform distribution with a range from 1 to R_{task} to generate one value for a vector τ. For each element of τ thus generated, the m iterations of the inner **for** loop (Line 3) generate one row of the ETC matrix. For the i-th iteration of the outer **for** loop, each iteration of the inner **for** loop produces one element of the ETC matrix by multiplying $\tau[i]$ with a random number sampled from a uniform distribution ranging from 1 to R_{mach}.

In the range-based ETC generation, it is possible to obtain high task heterogeneity low machine heterogeneity ETC matrices with characteristics similar to that of low task heterogeneity high machine heterogeneity ETC matrices if $R_{task} = R_{mach}$. In realistic HC systems, the variation that tasks show in their computational needs is generally larger than the variation that machines show in their capabilities. Therefore it is assumed here that requirements of high heterogeneity tasks are likely to be more "heterogeneous" than the capabilities of high heterogeneity machines (i.e., $R_{task} \gg R_{mach}$). However, for the ETC matrices generated here, low heterogeneity in both machines and tasks is assumed to be same. Table 1 shows typical values for R_{task} and R_{mach} for low and high heterogeneities. Tables 2 through 5 show four ETC matrices generated by the range-based method. The execution time values in Table 2 are

Table 1. Suggested values for R_{task} and R_{mach} for a realistic HC system for high heterogeneity and low heterogeneity.

	high	low
task	10^5	10^1
machine	10^2	10^1

much higher than the execution time values in Table 5. The difference in the values between these two tables would be reduced if the range for the low task heterogeneity was changed to 10^3 to 10^4 instead of 1 to 10.

With the range-based method, low task heterogeneity high machine heterogeneity ETC matrices tend to have high heterogeneity for both tasks and machines, due to method used for generation. For example, in Table 5, original τ vector values were selected from 1 to 10. When each entry is multiplied by a number from 1 to 100 for high machine heterogeneity this generates a task heterogeneity comparable to machine heterogeneity. It is shown later in Section 3.2 how to produce low task heterogeneity high machine heterogeneity ETC matrices which do show low task heterogeneity.

3.2. Coefficient-of-Variation Based ETC Matrix Generation

A modification of the procedure in Figure 1 defines the coefficient of variation, V, of execution time values as a measure of heterogeneity (instead of the range of execution time values). The coefficient of variation of a set of values is a better measure of the dispersion in the values than the standard deviation because it expresses the standard deviation as a percentage of the mean of the values [11]. Let σ and μ be the standard deviation and mean, respectively, of a set of execution time values. Then $V = \sigma/\mu$. The coefficient-of-variation-based ETC generation method provides a greater control over spread of the execution time values (i.e., heterogeneity) in any given row or column of the ETC matrix than the range-based method.

The coefficient-of-variation-based (CVB) ETC generation method works as follows. A task vector, q, of expected execution times with the desired task heterogeneity must be generated. Essentially, $q[i]$ is the execution time of task i on an "average" machine in the HC suite. For example, if the HC suite consists of an IBM SP/2, an Alpha server, and a Sun SPARC 5 workstation, then q would represent estimated execution times of the tasks on the Alpha server.

To generate q, two input parameters are needed: μ_{task} and V_{task}. The input parameter, μ_{task} is used to set the average of the values in q. The input parameter V_{task} is the desired coefficient of variation of the values in q. The value of V_{task} quantifies task heterogeneity, and is larger for higher task heterogeneity. Each element of the task vector q is then used to produce one row of the ETC matrix such that the desired coefficient of variation of values in each row is V_{mach}, another input parameter. The value of V_{mach} quantifies machine heterogeneity, and is larger for higher machine heterogeneity. Thus μ_{task}, V_{task}, and V_{mach} are the three input parameters for the CVB ETC generation method.

A direct approach to simulating HC environments should use the probability distribution that is empirically found to represent closely the distribution of task execution times. However, no standard benchmarks for HC systems are currently available. Therefore, this research uses a distribution which, though not necessarily reflective of an actual HC scenario, is flexible enough to be adapted to one. Such a distribution should not produce negative values of task execution times (e.g., ruling out Gaussian distribution), and should have a variable coefficient of variation (e.g., ruling out exponential distribution).

The gamma distribution is a good choice for the CVB ETC generation method because, with proper constraints on its characteristic parameters, it can approximate two other probability distributions, namely the Erlang-k and Gaussian (without the negative values) [11, 15]. The fact that it can approximate these two other distributions is helpful because this increases the chances that the simulated ETC matrices could be synthesized closer to some real life HC environment.

The uniform distribution can also be used but is not as flexible as the gamma distribution for two reasons: (1) it does not approximate any other distribution, and (2) the characteristic parameters of a uniform distribution cannot take all real values (explained later in the Section 3.3).

The gamma distribution [11, 15] is defined in terms of characteristic shape parameter, α, and scale parameter, β. The characteristic parameters of the gamma distribution can be fixed to generate different distributions. For example, if α is fixed to be an integer, then the gamma distribution becomes an Erlang-k distribution. If α is large enough, then the gamma distribution approaches a Gaussian distribution (but still does not return negative values for task execution times).

Figures 2(a) and 2(b) show how a gamma density function changes with the shape parameter α. When the shape parameter increases from two to eight, the shape of the distribution changes from a curve biased to the left to a more balanced bell-like curve. Figures 2(a), 2(c) and 2(d) show

Table 2. A high task heterogeneity low machine heterogeneity matrix generated by the range-based method using R_{task} and R_{mach} values of Table 1.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	333304	375636	198220	190694	395173	258818	376568
t_2	442658	400648	346423	181600	289558	323546	380792
t_3	75696	103564	438703	129944	67881	194194	425543
t_4	194421	392810	582168	248073	178060	267439	611144
t_5	466164	424736	503137	325183	193326	241520	506642
t_6	665071	687676	578668	919104	795367	390558	758117
t_7	177445	227254	72944	139111	236971	325137	347456
t_8	32584	55086	127709	51743	100393	196190	270979
t_9	311589	568804	148140	583456	209847	108797	270100
t_{10}	314271	113525	448233	201645	274328	248473	170176
t_{11}	272632	268320	264038	140247	110338	29620	69011
t_{12}	489327	393071	225777	71622	243056	445419	213477

Table 3. A high task heterogeneity high machine heterogeneity matrix generated by the range-based method using R_{task} and R_{mach} values of Table 1.

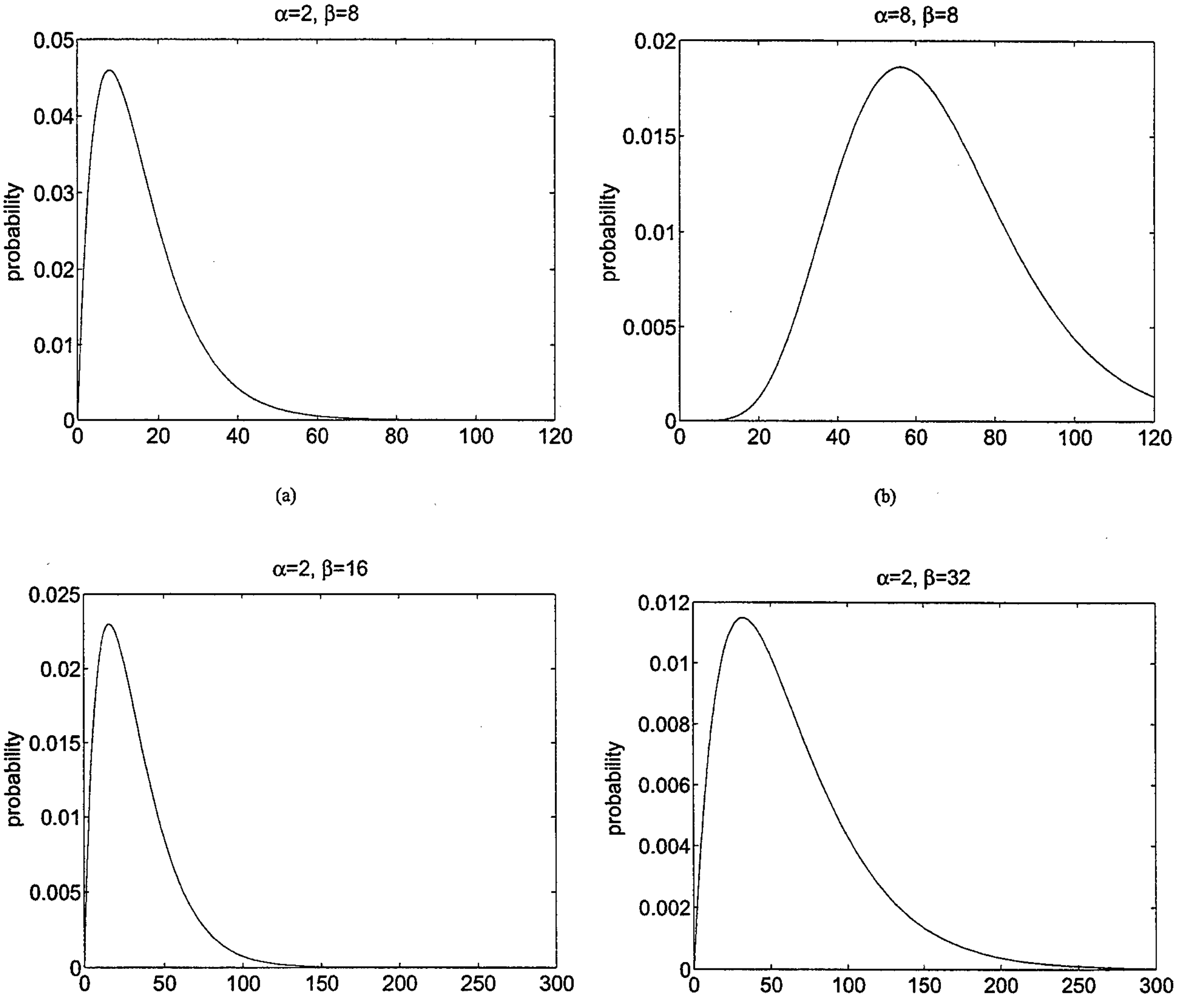
	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	2425808	3478227	719442	2378978	408142	2966676	2890219
t_2	2322703	2175934	228056	3456054	6717002	5122744	3660354
t_3	1254234	3182830	4408801	5347545	4582239	6124228	5343661
t_4	227811	419597	13972	297165	438317	23374	135871
t_5	6477669	5619369	707470	8380933	4693277	8496507	7279100
t_6	1113545	1642662	303302	244439	1280736	541067	792149
t_7	2860617	161413	2814518	2102684	8218122	7493882	2945193
t_8	1744479	623574	1516988	5518507	2023691	3527522	1181276
t_9	6274527	1022174	3303746	7318486	7274181	6957782	2145689
t_{10}	1025604	694016	169297	193669	1009294	1117123	690846
t_{11}	2390362	1552226	2955480	4198336	1641012	3072991	3262071
t_{12}	96699	882914	63054	199175	894968	248324	297691

Table 4. A low task heterogeneity low machine heterogeneity matrix generated by the range-based method using R_{task} and R_{mach} values of Table 1.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	22	21	6	16	15	24	13
t_2	7	46	5	28	45	43	31
t_3	64	83	45	23	58	50	38
t_4	53	56	26	42	53	9	58
t_5	11	12	14	7	8	3	14
t_6	33	31	46	25	23	39	10
t_7	24	11	17	14	25	35	4
t_8	20	17	23	4	3	18	20
t_9	13	28	14	7	34	6	29
t_{10}	2	5	7	7	6	3	7
t_{11}	16	37	23	22	23	12	44
t_{12}	8	66	47	11	47	55	56

Table 5. A low task heterogeneity high machine heterogeneity matrix generated by the range-based method using R_{task} and R_{mach} values of Table 1.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	440	762	319	532	151	652	308
t_2	459	205	457	92	92	379	60
t_3	499	263	92	152	75	18	128
t_4	421	362	347	194	241	481	391
t_5	276	636	136	355	338	324	255
t_6	89	139	37	67	9	53	139
t_7	404	521	54	295	257	208	539
t_8	49	114	279	22	93	39	36
t_9	59	35	184	262	145	287	277
t_{10}	7	235	44	81	330	56	78
t_{11}	716	601	75	689	299	144	457
t_{12}	435	208	256	330	6	394	419



(a)

(b)

(c)

(d)

Figure 2. Gamma probability density function for (a) $\alpha = 2$, $\beta = 8$, (b) $\alpha = 8$, $\beta = 8$, (c) $\alpha = 2$, $\beta = 16$, and (d) $\alpha = 2$, $\beta = 32$.

```
(1) $\alpha_{task} = 1/V_{task}^2$; $\alpha_{mach} = 1/V_{mach}^2$;
    $\beta_{task} = \mu_{task}/\alpha_{task}$
(2) for $i$ from 0 to $(t-1)$
(3)     $q[i] = G(\alpha_{task}, \beta_{task})$
        /* $q[i]$ will be used as mean
        of $i$-th row of ETC matrix */
(4)     $\beta_{mach}[i] = q[i]/\alpha_{mach}$
        /* scale parameter for $i$-th row */
(5)     for $j$ from 0 to $(m-1)$
(6)         $e[i,j] = G(\alpha_{mach}, \beta_{mach}[i])$
(7)     endfor
(8) endfor
```

Figure 3. The general CVB method for generating ETC matrices.

```
(1) $\alpha_{task} = 1/V_{task}^2$; $\alpha_{mach} = 1/V_{mach}^2$;
    $\beta_{mach} = \mu_{mach}/\alpha_{mach}$
(2) for $j$ from 0 to $(m-1)$
(3)     $p[j] = G(\alpha_{mach}, \beta_{mach})$
        /* $p[j]$ will be used as mean
        of $j$-th column of ETC matrix */
(4)     $\beta_{task}[j] = p[j]/\alpha_{task}$
        /* scale parameter for $j$-th column */
(5)     for $i$ from 0 to $(t-1)$
(6)         $e[i,j] = G(\alpha_{task}, \beta_{task}[j])$
(7)     endfor
(8) endfor
```

Figure 4. The CVB method for generating low task heterogeneity high machine heterogeneity ETC matrices.

the effect on the distribution caused by an increase in the scale parameter from 8 to 16 to 32. The two-fold increase in the scale parameter does not change the shape of the graph (the curve is still biased to the left); however the curve now has twice as large a domain (i.e., range on x-axis).

The gamma distribution's characteristic parameters, α and β, can be easily interpreted in terms of μ_{task}, V_{task}, and V_{mach}. For a gamma distribution, $\sigma = \beta\sqrt{\alpha}$, and $\mu = \beta\alpha$, so that $V = \sigma/\mu = 1/\sqrt{\alpha}$ (and $\alpha = 1/V^2$). Then $\alpha_{task} = 1/V_{task}^2$ and $\alpha_{mach} = 1/V_{mach}^2$. Further, because $\mu = \beta\alpha$, $\beta = \mu/\alpha$, and $\beta_{task} = \mu_{task}/\alpha_{task}$. Also, for task i, $\beta_{mach}[i] = q[i]/\alpha_{mach}$.

Let $G(\alpha, \beta)$ be a number sampled from a gamma distribution with the given parameters. (Each invocation of $G(\alpha, \beta)$ returns a new sample.) Figure 3 shows the general procedure for the CVB ETC generation.

Given the three input parameters, V_{task}, V_{mach}, and μ_{task}, Line (1) of Figure 3 determines the shape parameter α_{task} and scale parameter β_{task} of the gamma distribution that will be later sampled to build the task vector q. Line (1) also calculates the shape parameter α_{mach} to use later in Line (6). In the i-th iteration of the outer **for** loop (Line 2) in Figure 3, a gamma distribution with parameters α_{task} and β_{task} is sampled to obtain $q[i]$. Then $q[i]$ is used to determine the scale parameter $\beta_{mach}[i]$ (to be used later in Line (6)). For the i-th iteration of the outer **for** loop (Line 2), each iteration of the inner **for** loop (Line 5) produces one element of the i-th row of the ETC matrix by sampling a gamma distribution with parameters α_{mach} and $\beta_{mach}[i]$. One complete row of the ETC matrix is produced by m iterations of the inner **for** loop (Line 5). Note that while each row in the ETC matrix has gamma distributed execution times, the execution times in columns are not gamma distributed.

The ETC generation method of Figure 3 can be used to generate high task heterogeneity high machine heterogeneity ETC matrices, high task heterogeneity low machine heterogeneity ETC matrices, and low task heterogeneity low machine heterogeneity ETC matrices, but cannot generate low task heterogeneity high machine heterogeneity ETC matrices. To satisfy the heterogeneity quadrants of Section 2, each column in the final low task heterogeneity high machine heterogeneity ETC matrix should reflect the low task heterogeneity of the "parent" task vector q. This condition would not necessarily hold if rows of the ETC matrix were produced with a high machine heterogeneity from a task vector of low heterogeneity. This is because a given column may be formed from widely different execution time values from different rows because of the high machine heterogeneity. That is, any two entries in a given column are based on different values of $q[i]$ and α_{mach}, and may therefore show high task heterogeneity as opposed to the intended low task heterogeneity. In contrast, in a high task heterogeneity low machine heterogeneity ETC matrix the low heterogeneity among the machines for a given task (across a row) is based on the same $q[i]$ value.

One solution is to generate what is in effect a transpose of a high task heterogeneity low machine heterogeneity matrix to produce a low task heterogeneity high machine heterogeneity one. The transposition can be built into the procedure as shown in Figure 4. The procedure in Figure 4 is very similar to the one in Figure 3. The input parameter μ_{task} is replaced with μ_{mach}. Here, first a machine vector, p, (with an average value of μ_{mach}) is produced. Each element of this "parent" machine vector is then used to generate one low task heterogeneity column of the ETC matrix, such that the high machine heterogeneity present in p is reflected in all rows. This approach for generating low task heterogeneity high machine heterogeneity ETC matrices can also be used with the range-based method.

Tables 6 through 11 show some sample ETC matrices generated using the CVB ETC generation method. Tables 6 and 7 both show high task heterogeneity low machine heterogeneity ETC matrices. In both tables, the spread of the execution time values in columns is higher than that in rows. The ETC matrix in Table 7 has a higher task heterogeneity (higher V_{task}) than the ETC matrix in Table 6. This can be seen in a higher spread in the columns of matrix in Table 7 than that in Table 6.

Tables 8 and 9 show high task heterogeneity high machine heterogeneity and low task heterogeneity low machine heterogeneity ETC matrices, respectively. The execution times in Table 8 are widely spaced along both rows and columns. The spread of execution times in Table 9 is smaller along both columns and rows, because both V_{task} and V_{mach} are smaller.

Tables 10 and 11 show low task heterogeneity high machine heterogeneity ETC matrices. In both tables, the spread of the execution time values in rows is higher than that in columns. ETC matrix in Table 11 has a higher machine heterogeneity (higher V_{mach}) than the ETC matrix in Table 10. This can be seen in a higher spread in the rows of matrix in Table 11 than that in Table 10.

3.3. Uniform Distribution in the CVB Method

The uniform distribution could also be used for the CVB ETC generation method. The uniform distribution's characteristic parameters a (lower bound for the range of values) and b (upper bound for the range of values), can be easily interpreted in terms of μ_{task}, V_{task}, and V_{mach}. (Recall that $V_{task} = \sigma_{task}/\mu_{task}$ and $V_{mach} = \sigma_{mach}/\mu_{mach}$). For a uniform distribution, $\sigma = (b-a)/\sqrt{12}$ and $\mu = (b+a)/2$ [15]. So that

$$a + b = 2\mu \tag{1}$$

$$a - b = -\sigma\sqrt{12} \tag{2}$$

Adding Equations (1) and (2),

$$a = \mu - \sigma\sqrt{3} \tag{3}$$

$$a = \mu(1 - (\sigma/\mu)\sqrt{3}) \tag{4}$$

$$a = \mu(1 - V\sqrt{3}) \tag{5}$$

Also,

$$b = 2\mu - a \tag{6}$$

The Equations (5) and (6) can be used to generate the task vector q from the uniform distribution with the following parameters:

$$a_{task} = \mu_{task}(1 - V_{task}\sqrt{3}) \tag{7}$$

$$b_{task} = 2\mu_{task} - a_{task} \tag{8}$$

Once the task vector q has been generated, the i-th row of the ETC matrix can be generated by sampling (m times) a uniform distribution with the following parameters:

$$a_{mach} = q[i](1 - V_{mach}\sqrt{3}) \tag{9}$$

$$b_{mach} = 2q[i] - a_{mach} \tag{10}$$

The CVB ETC generation using the uniform distribution, however, places a restriction on the values of V_{task} and V_{mach}. Because both a_{task} and a_{mach} have to be positive, it follows from Equations (7) and (9) that the maximum value for V_{mach} or V_{task} is $1/\sqrt{3}$. Thus, for the CVB ETC generation, the gamma distribution is better than the uniform distribution because it does not restrict the values of task or machine heterogeneities.

3.4. Producing Consistent ETC Matrices

The procedures given in Figures 1, 3, and 4 produce inconsistent ETC matrices. Consistent ETC matrices can be obtained from the inconsistent ETC matrices generated above by sorting the execution times for each task on all machines (i.e., sorting the values within each row and doing this for all rows independently). From the inconsistent ETC matrices generated above, partially-consistent matrices consisting of an $i \times k$ sub-matrix could be generated by sorting the execution times across a random subset of k machines for each task in a random subset of i tasks.

It should be noted from Tables 10 and 11 that the greater the difference in machine and task heterogeneities, the higher the degree of consistency in the inconsistent low task heterogeneity high machine heterogeneity ETC matrices. For example, in Table 11 all tasks show consistent execution times on all machines except on the machines that correspond to columns 3 and 4. As mentioned in Section 1, these degrees and classes of mixed-machine heterogeneity can be used to characterize many different HC environments.

4. Analysis and Synthesis

Once the actual ETC matrices from a real life scenario are obtained, they can be analyzed to estimate the probability distribution of the execution times, and the values of the model parameters (i.e., V_{task}, V_{mach}, and μ_{task} (or μ_{mach}, if a low task heterogeneity high machine heterogeneity ETC matrix is desired)) appropriate for the given real life scenario. The above analysis could be carried out using common statistical procedures [9]. Once a model of a particular HC system is available, the effect of changes in the workload (i.e., the tasks arriving for service in the system) and the system (i.e., the machines present in the HC system) can be studied in a controlled manner by simply changing the parameters of the ETC model.

Table 6. A high task heterogeneity low machine heterogeneity matrix generated by the CVB method. $V_{task} = 0.3$, $V_{mach} = 0.1$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	628	633	748	558	743	684	740	692	593	554
t_2	688	712	874	743	854	851	701	701	811	864
t_3	965	1029	1087	1020	921	825	1238	934	928	1042
t_4	891	866	912	896	776	993	875	999	919	860
t_5	1844	1507	1353	1436	1677	1691	1508	1646	1789	1251
t_6	1261	1157	1193	1297	1261	1251	1156	1317	1189	1306
t_7	850	928	780	1017	761	900	998	838	797	824
t_8	1042	1291	1169	1562	1277	1431	1236	1092	1274	1305
t_9	1309	1305	1641	1225	1425	1280	1388	1268	1290	1549
t_{10}	881	865	752	893	883	813	892	805	873	915

Table 7. A high task heterogeneity low machine heterogeneity matrix generated by the CVB method. $V_{task} = 0.5$, $V_{mach} = 0.1$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	377	476	434	486	457	486	431	417	429	428
t_2	493	370	400	420	502	472	475	440	483	576
t_3	745	646	922	650	791	878	853	791	756	788
t_4	542	490	469	559	488	498	509	431	547	542
t_5	625	666	618	710	624	615	618	599	522	540
t_6	921	785	759	979	865	843	853	870	939	801
t_7	677	767	750	720	797	728	941	717	686	870
t_8	428	418	394	460	434	427	378	427	447	466
t_9	263	289	267	231	243	222	283	257	240	247
t_{10}	1182	1518	1272	1237	1349	1218	1344	1117	1122	1260
t_{11}	1455	1384	1694	1644	1562	1639	1776	1813	1488	1709
t_{12}	3255	2753	3289	3526	2391	2588	3849	3075	3664	3312

Table 8. A high task heterogeneity high machine heterogeneity matrix generated by the CVB method. $V_{task} = 0.6$, $V_{mach} = 0.6$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	1446	1110	666	883	1663	1458	653	1886	458	1265
t_2	1010	588	682	1255	3665	3455	1293	1747	1173	1638
t_3	1893	2798	1097	465	2413	1184	2119	1955	1316	2686
t_4	1014	1193	275	1010	1023	1282	559	1133	865	2258
t_5	170	444	500	408	790	528	232	303	301	480
t_6	1454	1106	901	793	1346	703	1215	490	537	1592
t_7	579	1041	852	1560	1983	1648	859	683	945	1713
t_8	2980	2114	417	3005	2900	3216	421	2854	1425	1631
t_9	252	519	196	352	958	355	720	168	668	1017
t_{10}	173	235	273	176	110	127	93	276	390	103
t_{11}	115	74	251	71	107	479	153	138	274	189
t_{12}	305	226	860	554	394	344	68	86	223	120

Table 9. A low task heterogeneity low machine heterogeneity matrix generated by the CVB method. $V_{task} = 0.1$, $V_{mach} = 0.1$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	985	1043	945	835	830	1087	1009	891	1066	1075
t_2	963	962	910	918	1078	1091	881	980	1009	981
t_3	782	837	968	960	790	800	947	1007	1115	845
t_4	999	953	892	986	958	1006	1039	1072	1090	1030
t_5	971	972	913	1030	891	873	898	994	1086	1122
t_6	1155	1065	800	1247	980	1103	1228	1062	1011	1005
t_7	1007	1191	964	860	1034	896	1185	932	1035	1019
t_8	1088	864	972	984	736	950	944	994	970	894
t_9	878	967	954	917	942	978	1046	1134	985	1032
t_{10}	1210	1120	1043	1093	1386	1097	1202	1004	1185	1226
t_{11}	910	958	1046	1062	952	1054	1020	1175	850	1060
t_{12}	930	935	908	1155	991	997	828	1062	886	831

Table 10. A low task heterogeneity high machine heterogeneity matrix generated by the CVB method. $V_{task} = 0.1$, $V_{mach} = 0.6$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	1679	876	1332	716	1186	1860	662	833	534	804
t_2	1767	766	1327	711	957	2061	625	626	642	800
t_3	1870	861	1411	932	1065	1562	625	976	556	842
t_4	1861	817	1218	865	1096	1660	587	767	736	822
t_5	1768	850	1465	764	1066	1585	663	863	579	757
t_6	1951	807	1177	914	939	1483	573	961	643	712
t_7	1312	697	1304	921	1005	1639	562	831	633	784
t_8	1665	849	1414	795	1162	1593	577	791	709	774
t_9	1618	753	1283	794	1153	1673	639	787	563	744
t_{10}	1576	964	1373	752	950	1726	699	836	633	764
t_{11}	1693	742	1454	758	961	1781	721	988	641	793
t_{12}	1863	823	1317	890	1137	1812	704	800	479	848

Table 11. A low task heterogeneity high machine heterogeneity matrix generated by the CVB method. $V_{task} = 0.1$, $V_{mach} = 2.0$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	4784	326	1620	1307	3301	10	103	4449	228	40
t_2	4315	276	1291	1863	3712	11	91	5255	200	47
t_3	6278	269	1493	1181	3186	12	93	4604	235	46
t_4	4945	294	1629	1429	2894	14	87	4724	231	45
t_5	5276	321	1532	1516	2679	12	102	4621	205	46
t_6	4946	293	1467	1609	2661	10	96	3991	255	39
t_7	4802	327	1317	1668	2982	10	90	5090	252	42
t_8	5381	365	1698	1384	3668	12	99	5133	242	38
t_9	5011	255	1491	1386	3061	10	94	3739	216	42
t_{10}	5228	296	1489	1515	3632	12	107	4682	203	38
t_{11}	5367	319	1332	1363	3393	12	72	4769	221	43
t_{12}	4621	258	1473	1501	3124	12	96	4091	199	44

This experimental set-up can then be used to find out which mapping heuristics are best suited for a given set of model parameters (i.e., V_{task}, V_{mach}, and μ_{task} (or μ_{mach})). This information can be stored in a "look-up table," so as to facilitate the choice of a mapping heuristic given a set of model parameters. The look-up table can be part of the toolbox in the mapper.

The ETC model of Section 2 assumes that the machine heterogeneity is the same for all tasks, i.e., different tasks show the same general variation in their execution times over different machines. In reality this may not be true; the variation in the execution times of one task on all machines may be very different from some other task. To model the "variation in machine heterogeneity" along different rows (i.e., for different tasks), another level of heterogeneity could be introduced. For example, in the CVB ETC generation, instead of having a fixed value for V_{mach} for all the tasks, the value of V_{mach} for a given task could be variable, e.g., it could be sampled from a probability distribution. Once again, the nature of the probability distribution and its parameters will need to be decided empirically.

5. Related Work

To the best of the authors' knowledge, there is currently no work presented in the open literature that addresses the problem of modeling of execution times of the tasks in an HC system (except the already discussed work [13]). However, below are presented two tangentially related works.

A detailed workload model for parallel machines has been given in [4]. However the model is not intended for HC systems in that the machine heterogeneity is not modeled. Task execution times are modeled but tasks are assumed to be running on multiple processing nodes, unlike the HC environment presented here where tasks run on single machines only.

A method for generating random task graphs is given in [17] as part of description of the simulation environment for the HC systems. The method proposed in [17] assumes that the computation cost of a task t_i, averaged over all the machines in the system, is available as $\overline{w_i}$. The method does provide for characterizing the differences in the execution times of a given task on different processors in the HC system (i.e., machine heterogeneity). The "range percentage" (β) of computation costs on processors roughly corresponds to the notion of machine heterogeneity as p-resented here. The execution time, e_{ij}, of task t_i on machine m_j is randomly selected from the range, $\overline{w_i} \times (1-\beta/2) \leq e_{ij} \leq \overline{w_i} \times (1+\beta/2)$. However, the method in [17] does not provide for describing the differences in the execution times of all the tasks on an "average" machine in the HC system. The method in [17] does not tell how the differences in the values of $\overline{w_i}$ for different machines will be modeled. That is, the method is [17] does not consider task heterogeneity. Further, the model in [17] does not take into account the consistency of the task execution times.

6. Conclusions

To describe different kinds of heterogeneous environments, an existing model based on the characteristics of the ETC matrix was presented. The three parameters of this model (task heterogeneity, machine heterogeneity, and consistency) can be changed to investigate the performance of mapping heuristics for different HC systems and different sets of tasks. An existing range-based method for quantifying heterogeneity was described, and a new coefficient-of-variation-based method was proposed. Corresponding procedures for generating the ETC matrices representing various heterogeneous environments were presented. Sample ETC matrices were provided for both ETC generation procedures. The coefficient-of-variation-based ETC generation method provides a greater control over the spread of values (i.e., heterogeneity) in any given row or column of the ETC matrix than the range-based method. This characterization of HC environments will allow a researcher to simulate different HC environments, and then evaluate the behavior of the mapping heuristics under different conditions of heterogeneity.

Acknowledgments: The authors thank Anthony A. Maciejewski, Tracy D. Braun, and Taylor Kidd for their valuable comments and suggestions.

References

[1] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance* Master's thesis, Department of Computer Science, Naval Postgraduate School, 1997 (D. Hensgen, Advisor).

[2] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, R. F. Freund, and D. Hensgen, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 15–29.

[3] H. G. Dietz, W. E. Cohen, and B. K. Grant, "Would y-ou run it here... or there? (AHS: Automatic Heterogeneous Supercomputing)," *1993 International Conference on Parallel Processing (ICPP '93)*, Vol. II, Aug. 1993, pp. 217–221.

[4] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds., Vol. 1459, Lecture Notes in Computer Science, Vol. 1459, Springer-Verlag, New York, NY, 1998, pp. 1–15.

[5] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184–199.

[6] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[7] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: The Management System for Heterogeneous Networks," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 184–198.

[8] M. A. Iverson, F. Özgüner, and G. J. Follen, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 99–111.

[9] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., New York, NY, 1991.

[10] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July-Sep. 1998, pp. 42–51.

[11] L. L. Lapin, *Probability and Statistics for Modern Engineering*, Second Edition, Waveland Press, Inc., Prospect Heights, IL, 1998.

[12] N. Lopez-Benitez and J.-Y. Hyon, "Simulation of task graph systems in heterogeneous computing environments," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 112–124.

[13] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,"*Journal of Parallel and Distributed Computing*, Special Issue on Software Support for Distributed Computing, Vol. 59, No. 2, pp. 107–131, Nov. 1999.

[14] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *Encyclopedia of Electrical and Electronics Engineering, Vol. 8*, J. G. Webster, ed., John Wiley, New York, NY, 1999, pp. 679–690.

[15] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, NY, 1984.

[16] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[17] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 3–14.

[18] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of execution times on heterogeneous supercomputer architecture," *1993 International Conference on Parallel Processing (ICPP '93)*, Vol. I, Aug. 1993, pp. 219–225.

Biographies

Shoukat Ali is pursuing a PhD degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a Research Assistant. His main research topic is dynamic mapping of meta-tasks in heterogeneous computing systems. He has held teaching positions at Aitchison College and Keynesian Institute of Management and Sciences, both in Lahore, Pakistan. He was also a Teaching Assistant at Purdue. Shoukat received his MS degree in electrical and electronic engineering from Purdue University in 1999, and his BS degree in electrical and electronic engineering from the University of Engineering and Technology, Lahore, Pakistan in 1996. His research interests include computer architecture, parallel computing, and heterogeneous computing.

Howard Jay Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received BS degrees in both electrical engineering and management from MIT, and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Muthucumaru Maheswaran is an Assistant Professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a BSc degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an MSEE degree in 1994 and a PhD degree in 1998, both from the School of Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an M-SEE student at Purdue University. His research interests include computer architecture, distributed computing, heterogeneous computing, Internet and world wide web systems, metacomputing, mobile programs, network computing, parallel computing, resource management systems for metacomputing, and scientific computing. He has authored or coauthored 15 technical papers in these and related areas. He is a member of the Eta Kappa Nu honorary society.

Debra Hensgen is a member of the Research and Evaluation Team at OpenTV in Mountain View, California. OpenTV produces middleware for set-top boxes in support of interactive television. She received her PhD in the area of Distributed Operating Systems from the University of Kentucky. Prior to moving to private industry, as an Associate Professor in the systems area, she worked with students and colleagues to design and develop tools and systems for resource management, network re-routing algorithms and systems that preserve quality of service guarantees, and visualization tools for performance debugging of parallel and distributed systems. She has published numerous papers concerning her contributions to the Concurra toolkit for automatically generating safe, efficient concurrent code, the Graze parallel processing performance debugger, the SAAM path information base, and the SmartNet and MSHN Resource Management Systems.

Sahra Ali is pursuing a PhD degree at the School of Electrical and Computer Engineering at Purdue University, where she is currently a Research Assistant. Her main research topic is the modeling of reliability in software intensive communication networks. She has also been working as a software developer for Cisco Systems since 1997. She was previously a Teaching Assistant and lab coordinator at Purdue. Sahra received her MS degree in electrical engineering from Purdue University in 1998, and her BS degree in electrical and electronic engineering from Sharif University of Technology, Tehran, Iran in 1995. Her research interests include testing, reliability and availability of communication networks, as well as multimedia.

The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions [*]

Robert Armstrong
Debra Hensgen
Taylor Kidd

Computer Science Department
Naval Postgraduate School
Monterey, CA 93940

Abstract

In this paper we study the performance of four mapping algorithms. The four algorithms include two naive ones: Opportunistic Load Balancing (OLB), and Limited Best Assignment (LBA), and two intelligent greedy algorithms: an O(nm) greedy algorithm, and an O(n^2m) greedy algorithm. All of these algorithms, except OLB, use expected run-times to assign jobs to machines. As expected run-times are rarely deterministic in modern networked and server based systems, we first use experimentation to determine some plausible run-time distributions. Using these distributions, we next execute simulations to determine how the mapping algorithms perform. Performance comparisons show that the greedy algorithms produce schedules that, when executed, perform better than naive algorithms, even though the exact run-times are not available to the schedulers. We conclude that the use of intelligent mapping algorithms is beneficial, even when the expected time for completion of a job is not deterministic.

1 Introduction

This paper describes the experiments and simulations that we executed to determine the relative performance of certain mapping algorithms in different heterogeneous environments. In this paper we assume that all jobs are independent of one another. That is, they do not communicate or synchronize with one another. This type of architecture is common in today's LAN-based distributed server environment.

Our goal was to determine whether using intelligent mapping algorithms would be beneficial, even if the jobs did not run for exactly the amount of time expected. Intelligent mapping algorithms utilize the expected run-times of each job on each different machine to attempt to minimize some scalar performance metric. For our experiments, this metric is the time at which the last job completes. In particular, we were concerned about whether it would still be beneficial to use intelligent mapping if one or several jobs run for a substantially different amount of time than expected, but are still accurately characterized statistically. Because determining a perfect mapping is an NP-complete problem, we examined the performance of several different (polynomial) heuristics. The algorithms we chose are listed below.

- A naive O(n) algorithm known as Opportunistic Load Balancing (OLB). This algorithm simply places each job, in order of arrival, on the next available machine.
- A simple O(nm) algorithm known as Limited Best Assignment (LBA). This algorithm uses the expected run-time of each job on each machine. It assigns each job to the machine on which it has the least expected run-time, ignoring any other loads on the machines, including that produced by the jobs that it has assigned.

  This algorithm, though easily implementable in a scheduling framework that automatically assigns jobs to machines, is very similar to the algorithm used by many users who remotely start their jobs by hand at supercomputer centers without examining queue lengths.
- Two greedy algorithms, one of order O(nm) and the other of order O(n^2m). Both of these algorithms make use of the expected run-time of each job on each machine as well as the expected loads on each machine. These algorithms will be more fully described in Section 2.

[*] This research was supported by DARPA under contract number E583. Additional support was provided by the Naval Postgraduate School and the Institute for Joint Warfare Analysis.

The primary reasons for our study are both that jobs rarely execute for exactly the expected run-time and often the expected run-times are not exactly known. In a system where each job has exclusive use of a machine, differences between actual and predicted run-times occur either because (1) all of the compute characteristics [10] are not known or enumerated by the designer of the program, or (2) because the time to access memory and disk is stochastic and not deterministic. Of course, in many environments, additional non-determinism is due to other jobs running on the machine or simultaneously using a shared network or a shared file server. This paper focuses on those cases where one or more of the jobs being scheduled have run-times that could differ substantially from the expected run-time. For those cases, we seek to determine whether there is still an advantage to using an algorithm that makes use of expected run-times or whether a computationally simpler algorithm that does not require estimating run-times, such as Opportunistic Load Balancing (OLB), might not yield equivalently good performance.

In the next section, we describe the two greedy algorithms that we used in our experiments and simulations. We then describe our experiments concerning the non-determinism of expected run-times and examine, using the derived distributions in simulations, the performance of the intelligent algorithms. That is, we collect run-times for various jobs on various machines, analyze their distributions, and extrapolate these distributions for use in our simulations. We conclude the paper with a short summary and comparison to related work.

2 The Greedy Algorithms

In addition to the simple OLB and LBA algorithms described in the previous section, our experiments used two greedy algorithms. We now describe those algorithms in detail.

The first algorithm is an $O(nm)$ algorithm, where n is the number of jobs and m is the number of machines, and the second algorithm is of order $O(n^2m)$. Each algorithm first estimates the expected run-time of each job on each machine, assuming that if a job cannot execute on a particular machine, the estimation will be set to some very large number. As we describe these algorithms we will consider these expected run-times as elements of a 2-dimensional, n by m matrix called A. That is, $A[i,j]$ is the expected run-time of job i on machine j.

The $O(nm)$ algorithm, which, like in the SmartNet documentation [6], we will call Fast Greedy, considers the jobs in the order requested[1]. It first determines the value $A_{1,j}$, such that $A_{1,j} \leq A_{1,k}\ \forall\ k \in \{1..m\}$. It then assigns job 1 to machine j. Following this, it adds $A_{1,j}$ to all $A_{i,j}\ \forall\ i \in \{2..n\}$. Then, for each remaining job, $p \in \{2..m\}$, it determines the value $A_{p,j}$, such that $A_{p,j} \leq A_{p,k}\ \forall\ k \in \{1..m\}$. It then assigns job p to machine j. Following this, it adds $A_{p,j}$ to all $A_{i,j}\ \forall\ i \in \{p+1..n\}$. At each step, then, it is assigning each job to its best machine, given the previous assignments. We note that the jobs are assigned in the order in which they were requested.

The $O(n^2m)$ algorithm, which again borrowing from SmartNet nomenclature we call simply Greedy, actually computes two mappings using two different sub-algorithms and then chooses the mapping that gives the smallest sum of the predicted run-times, minimized over all machines. The two sub-algorithms are similar to the first greedy algorithm above, differing only in the order in which they assign jobs to machines. We first enumerate the steps of the first sub-algorithm.

1. Initialize the set $\{RemainingJobs\}$ to contain all jobs.
2. $\forall\ i \in \{RemainingJobs\}$, find $A_{i,j} \leq A_{i,k}\ \forall\ k \in \{Machines\}$. Call such an $A_{i,j}$, A_{i,min_i}.
3. Determine p such that $A_{p,min_p} \leq A_{i,min_i}\ \forall\ i \in \{RemainingJobs\}$.
4. Remove p from $\{RemainingJobs\}$, scheduling job p on machine min_p.
5. Add A_{p,min_p} to $A_{i,min_p}\ \forall\ i \in \{RemainingJobs\}$.
6. If $\{RemainingJobs\}$ is not empty, return to step 2.

The idea behind this first sub-algorithm is that, at each step, we attempt to minimize the time at which the last job, which has been thus far scheduled, finishes.

The second sub-algorithm differs from the first sub-algorithm in that, at the third step, it finds p such that $A_{p,min_p} \geq A_{i,min_i}\ \forall\ i \in \{RemainingJobs\}$. This algorithm, then tries to minimize the worst case time at each step.

3 Effect of Non-Determinism on Algorithm Performance

We now examine the effect of non-determinism on the performance of the greedy and LBA algorithms that we described above. Our reason for studying this

[1] In describing these algorithms, we use the term *order requested* to mean the order in which the job requests have been placed prior to invocation of the algorithm. We also investigated the performance of these algorithms if jobs are first sorted before these algorithms are invoked.

is because both the LBA and the greedy algorithms use the expected run-time to produce their mappings. One of our major motivations for this work is to determine whether such intelligent algorithms are still useful if the actual run-time is non-deterministic, that is, essentially sampled from a distribution around the expected run-time. In order to determine what distributions we should sample our run-times from in our simulation, we first conducted some experiments with actual programs to try to determine what types of distributions characterize their run-times.

3.1 Job Run-time Distributions

We have already explained why job-machine run-times are typically not constant, but rather vary according to some distribution. To test the performance of our algorithms, it is essential to draw samples of the run-times of jobs from a particular distribution; but first we need to determine some realistic distributions that we can use in our simulations. Therefore, we repeatedly executed some parallel and sequential programs, gathered run-time statistics, and analyzed them.

We performed several experiments using the NAS Benchmarks [3]. These benchmarks were used to determine the types of run-time distributions that may be typical for at least some jobs on some machines. We needed to determine sample parameters for these run-time distributions so that they could be reproduced by our simulator. While performing our tests, we controlled the following environmental characteristics: server location, network and server load, number of processors, amount of memory, and processor speed. Table 1 summarizes the configurations of our machines `caesar` and `elvis` upon which we ran our experiments.

	caesar	elvis
Type SGI	Challenge L	Onyx
Proc Speed (MHz)	200	150
Proc Type (MIPS)	R4400	R4400
# of Processors	4	4
Memory (Mbytes)	64	192
Secondary Unified Cache	4 Mb	1 Mb

Table 1: Configuration of SGI machines `caesar` and `elvis`, both running IRIX64 v6.2.

The jobs that we used throughout these experiments were from two sources: NASA's reference implementation for some of the NAS Benchmarks, and our own implementations of other NAS Benchmarks that met the required criteria. Four of the experiments use some version of the NAS Integer Sort (IS) Benchmark, implemented either in parallel on four processors, or in single processor mode. Two other experiments used the NAS Embarrassingly Parallel (EP) Benchmark run on a single processor. We now explain our experiments and their results.

3.1.1 Integer Sort, Executed on Four Processors

This experiment examined the run-time distribution of a version of the NAS Integer Sort Benchmark executed on four processors. We implemented the integer sort using a counting sort [5, pages 175–178] algorithm. We used Silicon Graphic's light weight process (thread) support functions, including `mfork()`, to implement our version of this benchmark.

We ran this sort across a heavily loaded network, obtaining both the executable and the data from a file server that was also heavily loaded. When run on `caesar`, the run-time distribution, for 100 executions, appears Gaussian.[2] Figure 1 shows a histogram of this distribution. When run on `elvis`, the run-time distribution, again for 100 executions, appears exponential and is shown in Figure 2. We note that the origin of the exponential distribution shown in Figure 2 is translated to approximately 3.0. That means that the sort had to run for *at least* 3.0 seconds before stopping. The distribution that we see very closely matches an exponential distribution with a mean of around 0.20, translated 3.0 seconds to the right. We expect that many jobs would have a distribution similar to this, because all jobs must run at least some amount of time[3].

In these experiments, we also see that memory size, and so, the need to swap to local disk, can have a definite effect upon the run-time distribution of a job. The integer sort on `elvis` completes, on average, 30% sooner than the same job on `caesar`. We note that, in this case, the amount of memory has more influence

[2] The form of the distributions were determined by carefully selecting the bin size and then curve fitting. The authors are familar with both visual and analytical tests for normality, but analytical tests were not used given the strong visual similarity of the frequency plots to that of a Normal curve. (The fact that some sample point frequencies lie above and below the selected Normal distribution is due to the number of samples being finite. Such phenomena would have appeared even if 100 data points had been sampled from a known Normal run-time distribution.)

[3] An exponential distribution is defined to start at 0.0. If applied, without translation, in this case, that would mean there is a strong possibility of near-zero run-times.

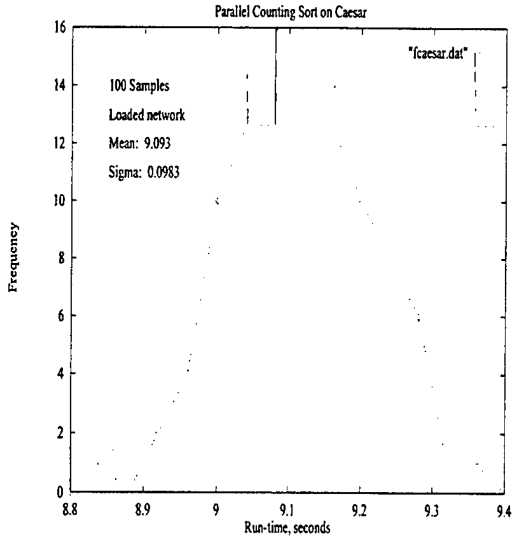


Figure 1: Forked counting sort, caesar.

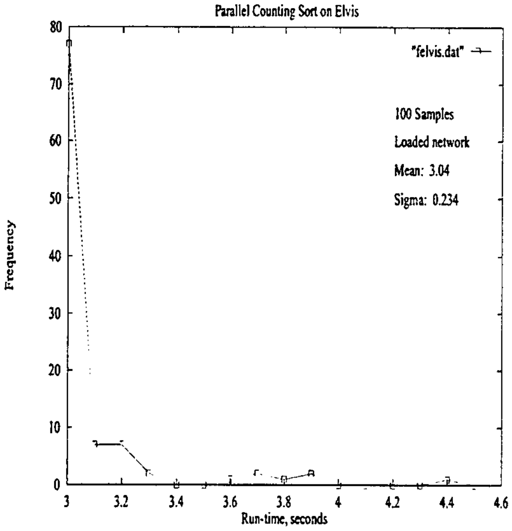


Figure 2: Forked counting sort, elvis.

on the run-time of the job than does the speed of the processor. Of primary importance, however, is the observation indicating that the same job, running on two different machines, not only has different mean run-times, but the distribution of run-times is different, yielding a Gaussian-like distribution on one machine and an exponential-like distribution on the other.

3.1.2 Integer Sort, Single Processor

This experiment is the same as that discussed in the last section, with the exception of being run on a single processor instead of being distributed across four processors. Although a slightly different C++ implementation was used, we again based our program on the counting sort.

When the integer sort was run on caesar and elvis, the run-time distribution was not easily characterized; however, it appears related to a Gaussian distribution. Histograms of the distributions, similar to that shown in Figure 4, are possibly multimodal, which indicates that multiple distributions may be present. While this experiment does not provide us with definitive results, it does point to the fact that run-time distributions can be quite complex. We suspect that these conditions are related to changes in the network and server loads.

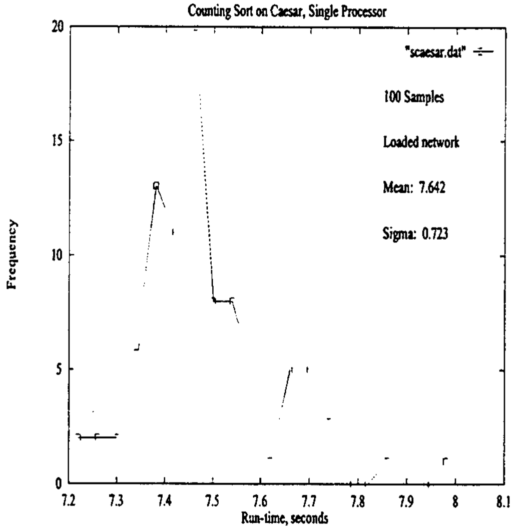


Figure 3: Counting sort, caesar, single processor.

Once again, this set of experiments showed us that additional memory can greatly enhance run-time performance. The tests on elvis ran 7 times faster than those run on caesar, which has the faster processors. The tests also show that run-time distributions can be very complex, and may be difficult to reproduce in a simulation. Although our simulations did not use such complex distributions, they should be modeled in future work.

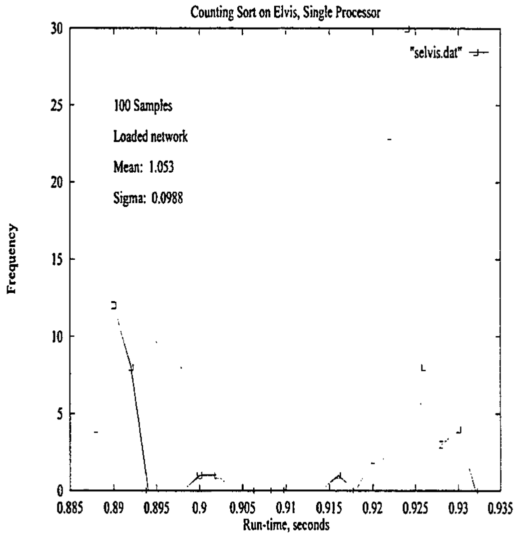


Figure 4: Counting sort, `elvis`, single processor.

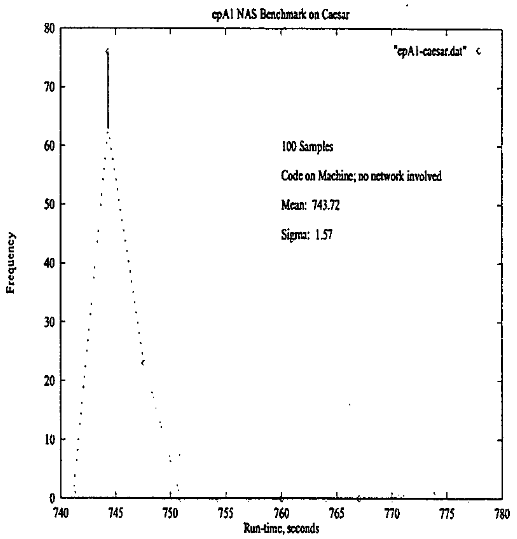


Figure 5: epA1 NAS Benchmark, with executable residing on local disk.

3.1.3 Embarrassingly Parallel NAS Benchmark

The next set of experiments that we describe compared the run-time distributions of compute intensive jobs run from local disk to those run across the network from a file server. The tests that we describe in this section were executed only on `caesar` because `elvis` did not have a sufficiently large local disk available. We used the reference implementation [3], from NASA, of the NAS Embarrassingly Parallel (EP) Benchmark. This implementation uses the portable Message Passing Interface (MPI) [12] to parallelize the code. The tests we ran, however, were compiled to be executed on a single processor[4]. The EP Benchmark was run 100 times for each test. See Figures 5 and 6.

3.2 Simulation Experiments

We now describe our simulation experiments that are aimed at examining how well the mapping algorithms performed when the jobs scheduled did not execute for exactly the mean run-time. The matrices that we refer to in the description below have rows indexed by the job and columns indexed by the machine.

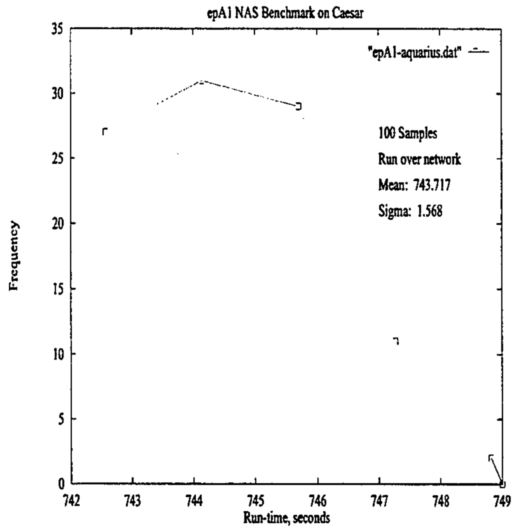


Figure 6: epA1 NAS Benchmark, files obtained over a lightly loaded network.

- Matrix Format. We used different matrices containing jobs and machines of varying characteristics. Each matrix contained mean run-times for each of five different jobs on each of ten different machines. The average means of the corresponding columns and rows were the same for all matrices and the jobs themselves were quite heterogeneous.

- Job Request Sets. In order to obtain different results for each matrix, we generated two random sequences of 125 job requests, which we will call 125-1 and 125-2, where each individual request was chosen according to a uniform random distribution from among five different jobs. We also generated two more ran-

[4] The MPI mechanism is still utilized in the EP Benchmark when it is compiled for a single processor.

dom sets, this time of 500 job requests, calling them 500–3 and 500–4. We did this to look at performance variations between job request orderings, as well as to examine any performance differences that might occur because fewer or more jobs were requested.

- Job Request Format. We generated each of the 5 jobs, for each request, at random. Thus, in these experiments, the jobs were requested in random order. This was done because the order of job request affects the schedule. The Fast Greedy Algorithm maps and schedules the jobs on machines in the order in which they are submitted. The Greedy Algorithm uses the order to break ties. We chose to execute these randomly ordered requests both because they more closely mimic a real environment where different jobs are submitted by different users and because we wished to examine whether these algorithms performed better or worse when unsorted, as opposed to sorted, requests were submitted.
- Run-time Generation for Simulations. We executed each simulation 15 times. In each run, a different value was used to seed the random number generator that was used to generate the simulated "actual" run-time duration. The total time required to execute each schedule was summed and the average was computed. Multiple seeds were used to ensure that our results were not skewed[5].
- Baseline Calculations. In addition to simulations where we generated simulated run-times from particular distributions, we performed some **baseline calculations**. These baseline calculations provided results that were, in effect, equivalent to running the simulation where the run-time of a job on a given machine was always exactly its expected run-time.
- Actual Run-time Distributions. When we generated run-times that were different from the mean predicted run-times, we ran experiments for both Gaussian and exponential distributions. Based upon our experiments with the NAS IS and EP Benchmarks above, we chose to implement a translated exponential distribution.

  Again, based upon our earlier experiments described in Section 3.1, we chose to use a truncated Gaussian distribution in our simulation experiments to mimic the Gamma distribution that best fit our data. We chose to truncate left of the mean at $\mu - \sigma$.

3.3 Results of Simulation Experiments where Jobs Ran for Times Different from the Predicted Run-times

This set of experiments examined the performance of intelligent mapping algorithms when job run-times differed from the expected run-times that were used to develop the mappings. Using the distributions identified in the previous experiments, we instantiated specific parameters in order to simulate some typical jobs. We simulated jobs with both exponential and truncated Gaussian run-time distributions. In this paper we summarize results; individual results from additional individual experiments, which are consistent with the conclusions that we make in this paper, can be found in Armstrong's thesis [2].

The graphs in this section compare the final completion times of the jobs under the various mappings. We use the label **Baseline** to mean that the value represented would be the completion time if all of the jobs ran for exactly their predicted mean run-times. In order to emphasize the differences between the values that we plot in the graph, we do not include the OLB run-times. The OLB run-times, for the exponential and Gaussian distribution simulations that we discuss below, averaged around 10,000 seconds in all cases shown, i.e., 500 requests.

3.3.1 Exponential Distribution Experiments

The results of these experiments compare the performance of the various mapping algorithms when all jobs have an exponential run-time distribution. We recall that the sample run-times from those experiments closely fit a shifted exponential distribution with mean of 3.0.

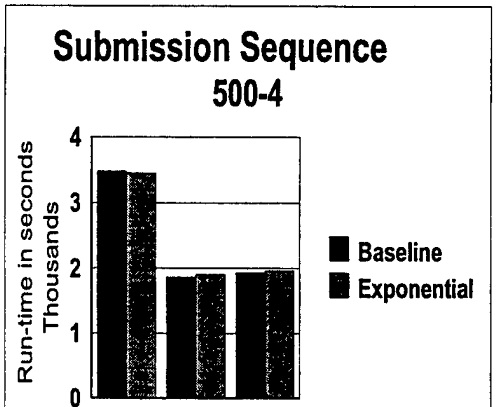


Figure 7: Exponential run-time distribution results, 500-4.

[5] This is a common method to reduce the influence of a single random number generation sequence that may be biased.

We now compare the time at which the last job finishes if executed according to each of the mappings, assuming that a job is not started on a machine until the last job completes. The figures in this section show both the expected completion time assuming deterministic run-times as well as under the assumption that the run-times are exponentially distributed, shifted to the right such that its mean matches the expected run-time.

Figure 7 shows these comparisons for some matrices that we used in our simulations. This figure shows that the schedules built by the intelligent mapping algorithms are still effective even though the actual run-time of a given job on a given machine can differ greatly from its expected run-time.

3.3.2 Truncated Gaussian Experiments

We then performed additional simulations to examine the performance of the the intelligent mapping algorithms when all jobs had approximately Gamma run-time distributions. We determined from our experiments that we could approximate such a distribution by truncating a Gaussian distribution to the left of the mean at roughly $\mu - \sigma$. Throughout this experiment, the mean, μ, was the expected run-time for the individual job/machine pair, and σ^2 was set to 300% of μ. Therefore, these experiments are useful in determining whether, when the variance is very large for all jobs, the greedy algorithms still performed much better than both the LBA and OLB algorithms. No negative run-times were generated in our experiments because the truncation value was always positive.

The results in Figure 8 show that the schedules are finishing up to 25% later than in the previous experiments. This not unexpected, as truncation will shift the mean of the resulting distribution to the right. In the next section we provide a theoretical discussion as to why we would expect the times to be at least 20% later. The results also show that the greedy algorithms still perform better than the OLB and LBA algorithms when job run-time distributions are truncated Gaussian with very large variances. Our experiments, and the theoretical explanation below, imply that it may be worthwhile to update the mapping as the jobs are being executed, to minimize the effect of the large job variances.

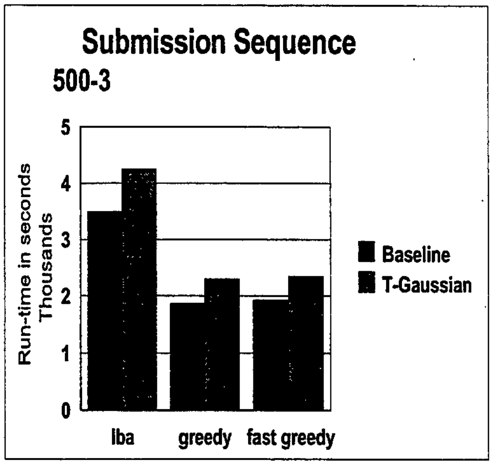


Figure 8: Truncated Gaussian run-time distribution results, 500-4.

3.3.3 Theoretical Explanation for Longer Run-times shown in Gaussian Experiments

To theoretically predict the new mean of the truncated distribution described in the last section, we can use simple Gaussian statistics [1]. Without loss of generality, our explanation uses a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. If $A(z_1)$ is the area under the distribution from the mean, $z = 0$, to $z = z_1$, then it can be easily shown that the new mean, μ_{new}, for our truncated distribution is

$$\mu_{new} = A^{-1}\left[\frac{.5 - A(1)}{2}\right] \qquad (1)$$

Using this, we see that the new mean should be $\mu_{new} = .20\sigma$.

Unfortunately, the truncation of the Gaussian distribution only accounts for a 20% increase in the mean. Therefore, this explanation alone leaves some 5% unaccounted for. The remaining 5% is due to two factors. The first can be traced to the fact that we are using a truncated Gaussian instead of a Gamma distribution. The second is the fact that the expected value of the maximum of several Gaussian distributions is not the maximum of the expected values. The application of this well-known probability result to quality of service metrics is documented elsewhere [9].

3.3.4 Comparison of the Two Greedy Algorithms

We note that in our results, presented both here and in Armstrong's thesis, the Greedy and Fast Greedy algorithms appeared to perform similarly. Over all of our experiments we only saw the Greedy Algorithm performing up to 15% better than the Fast Greedy Algorithm. Other work has suggested that the improvement should be much higher. However, the other work, to our knowledge, was based upon presenting *sorted* requests to these mapping algorithms. The theoretical explanation for these results is beyond the scope of this paper and is discussed in another paper [7].

4 Related Work

To our knowledge, no one else has studied the performance of intelligent heterogeneous mapping algorithms when the run-times of jobs are non-deterministic, by using the distributions of run-times for actual programs determined under different resource loadings.

Ibarra and Kim [8] were the first to study the performance of the algorithms upon which we concentrated. Their analytical study centered around determining the worst-case performance of the algorithms. Weissman [15] used simulation to study interference-based policies; that is, policies that take into account the fact that as you increase the load on any shared resource, the rate of execution of other jobs decreases. Our policies, and simulations, assumed that the jobs were executed on a first-come, first-served basis. Although we did not study their performance here, genetic algorithms have been proposed as a good way to schedule tasks on heterogeneous resources, particularly when communication or synchronization is needed between tasks [13], [14]. Many systems have followed the lead of SmartNet [6] in implementing intelligent schedulers, such as those we describe here, in their resource management systems [11], [4], [16].

5 Summary

In this paper, we experimented with several applications on resources with differing loads and fitted their run-times to distributions. We then used these distributions to determine via simulation whether, when the run-times are non-deterministic, it is still beneficial to use intelligent algorithms that make use of the expected run-times to compute a mapping. We found that it continues to be beneficial even when the expected run-time distributions have large variances. As the distributions in our simulations were derived from the execution of actual programs, our distributions are realistic. However, there are additional distributions that are also realistic that we have not yet examined. We intend to pursue these in future work.

References

[1] Alder, H. L., and Roessler, E. B. *Introduction to Probability and Statistics*, third ed. Freeman, London, England, 1964.

[2] Armstrong, R. K. Investigation of Effect of Different Run-time Distributions on SmartNet Performance. Master's thesis, U.S. Naval Postgraduate School, September 1997.

[3] Bailey, D., et al. The NAS Parallel Benchmarks 2.0. Tech. Rep. NAS-95-020, NASA Ames Research Center, December 1995.

[4] Beguelin, A., et al. *HeNCE: A User' Guide.* Oak Ridge National Laboratory and University of Tennessee, December 1992. The document itself is available on the web at cs.utk.edu.

[5] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 1990.

[6] Freund, R., Kidd, T., Hensgen, D., and Moore, L. Smartnet: A Scheduling Framework for Heterogeneous Computing. *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks* (1996).

[7] Hensgen, D., Kidd, T., and Armstrong, R. Comparison of greedy algorithms for scheduling jobs in a heterogeneous environments. In progress.

[8] Ibarra, and Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM* (1977).

[9] Kidd, T., and Hensgen, D. Why the mean is inadequate for accurate scheduling decisions. In progress.

[10] Kidd, T., Hensgen, D., Freund, R., Kussow, M., and Campbell, M. Compute Characteristics: A Useful Characterization of Job Runtimes. In preparation for submission (1998).

[11] Neuman, B. C., and Rao, S. The Prospero Resource Manager: A Scalable Framework for Processor Allocation in Distributed Systems. *Concurrency: Practice and Experience* (1994).

[12] Pacheco, P. A User's Guide to MPI. Tech. rep., Department of Mathematics, University of San Francisco, March 1995.

[13] Singh, H., and Youssef, A. Mapping and Scheduling Heterogeneous Task Graphs using Genetic Algorithms. *Proceedings of the Heterogeneous Computing Workshop* (1996).

[14] Wang, L., Siegel, H. J., and Roychowdhury, V. P. A Genetic-Algorithm-Based Approach for Task Matching and Scheduling in Heterogeneous Computing Environments. *Proceedings of the Heterogeneous Computing Workshop* (1996).

[15] Weissman, J. B. The Interference Paradigm for Network Job Scheduling. *Proceedings of the Heterogeneous Computing Workshop* (1996).

[16] Zhou, Zheng, Wang, and Delisle. Utopia: A load sharing facility for lage heterogeneous distributed computer systems. *Software: Practice and Experience* (1993).

Biographies

Major Robert K. Armstrong is currently in charge of the Modeling and Simulation Laboratory for the Marine Corps Air Ground Combat Center, Twentynine Palms, California. He received his BS in Engineering from the United States Naval Academy in 1985, is a graduate of the Amphibious Warfare School in Quantico, Virginia, and has earned an MS in Computer Science from the Naval Postgraduate School, Monterey, California in 1997. Major Armstrong has served in the capacity of Artillery Officer with the 1st Marine Division in Korea, Somalia, and Kuwait. His interests include computer architecture, distributed systems, and modeling and simulation for training.

Debra Hensgen received her Ph.D. in Computer Science, in the area of Distributed Operating Systems from the University of Kentucky in 1989. She is currently an Associate Professor of Computer Science at the Naval Postgraduate School in Monterey, California. She moved to Monterey from the University of Cincinnati three years ago where she was first appointed as an Assistant Professor and then a tenured Associate Professor of Electrical and Computer Engineering. Her research interests include resource management and allocation systems and tools for concurrent programming. She has authored numerous papers in these areas. She is currently a Subject Area Editor for the Journal of Parallel and Distributed Computing and is the chief architect and a co-Principal Investigator for the DARPA-funded MSHN project which is part of DARPA's larger QUORUM program.

Taylor Kidd is an Associate Professor of Computer Science at the Naval Postgraduate School (NPS) in Monterey, California. He received his Ph.D. in Electrical and Computer Engineering from the University of California at San Diego (UCSD) in 1991. He received his MS and BS, also in Electrical and Computer Engineering, from UCSD in 1986 and 1985 respectively. Prior to accepting a position at the NPS, he was a researcher at the Navy's NRaD laboratory in San Diego, California. His current interests include distributed computing and the application of stochastic filtering and estimation theory to distributed systems. He is a co-Principal Investigator, along with Debra Hensgen, for the DARPA-funded MSHN project which is part of DARPA's larger QUORUM program.

Adaptive Communication Algorithms for Distributed Heterogeneous Systems

Prashanth B. Bhat * and Viktor K. Prasanna *
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
{prabhat, prasanna}@halcyon.usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
raghu@aero.org

Abstract

Heterogeneous network-based systems are emerging as attractive computing platforms for HPC applications. This paper discusses fundamental research issues that must be addressed to enable network-aware communication at the application level. We present a uniform framework for developing adaptive communication schedules for various collective communication patterns. Schedules are developed at run-time, based on network performance information obtained from a directory service. We illustrate our framework by developing communication schedules for total exchange. Our first algorithm develops a schedule by computing a series of matchings in a bipartite graph. We also present a $O(P^3)$ heuristic algorithm, whose completion time is within twice the optimal. This algorithm is based on the open shop scheduling problem. Simulation results show performance improvements of a factor of 5 over well known homogeneous scheduling techniques.

1. Introduction

With recent advances in high-speed networks, *metacomputing* has emerged as a viable and attractive computational paradigm. A metacomputing system [23] consists of geographically distributed supercomputers and visualization devices. These are interconnected by a heterogeneous collection of local and wide-area networks. High performance applications can be executed over such a *networked virtual supercomputer*, wherein the distributed computational resources are used in a coordinated way, very much as though they were part of a single computer system.

The potential of metacomputing has been demonstrated by the Global Information Infrastructure (GII) testbed at SC '95 [16]. The testbed linked dozens of high performance computers and visualization machines with existing high-bandwidth networks and telephone systems. Some of the leading metacomputing research projects are Globus, Legion, VDCE, and MSHN, to name a few. These will be discussed further in Section 2.

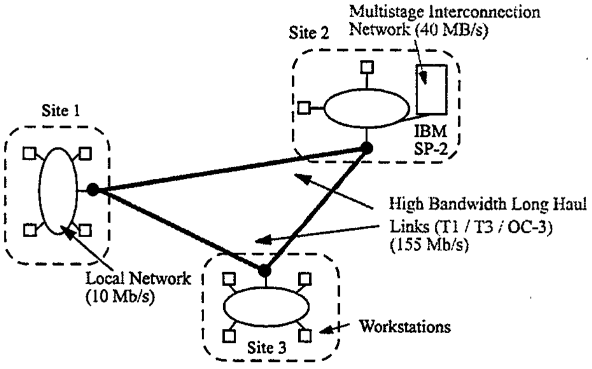

Multistage Interconnection Network (40 MB/s)
Site 2
Site 1
IBM SP-2
High Bandwidth Long Haul Links (T1 / T3 / OC-3) (155 Mb/s)
Local Network (10 Mb/s)
Workstations
Site 3

Figure 1. A typical metacomputing system.

Figure 1 shows an example of a small-scale metacomputing system. The compute nodes in the system are located at three different sites. Some of the nodes are high-end supercomputing systems, while others are workstations. The example shown in this figure has three kinds of interconnection networks: (i) the multi-stage interconnection network within the IBM SP-2, (ii) local networks at each site, and (iii) high bandwidth long haul ATM or T3 links between the sites.

Although network-based computing platforms offer significant advantages for high performance computing, effective use of their resources is still a major challenge. Computational and communication resources are typically shared among different applications. Computational tasks may be preempted by higher priority processes. Network conditions change continuously, and run-time loads cannot be determined apriori. Applications must therefore be capable of

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

adapting to changing system conditions.

In this paper, we develop communication techniques that enable applications to adapt to variations in network conditions. We focus on collective communication patterns among application processes executing over a heterogeneous network. Our goal is to develop efficient application-level implementations of these communication routines. We assume the availability of end-to-end send and receive communication routines, which can be invoked between any pair of nodes. The details of network topology, routing, and flow control policies are therefore hidden from the application.

Efficient algorithms for various collective communication patterns have been developed for tightly coupled parallel architectures with homogeneous networks [2, 19]. These algorithms have been incorporated into communication libraries and implementations of MPI. However, these techniques can perform poorly in metacomputing systems due to the heterogeneity among network bandwidths. Further, these are static algorithms, with no provision for adaptivity to network conditions.

In Section 3, we introduce our approach for developing network-aware communication techniques. The key components of our approach are: (i) a directory service which provides information on current network performance, (ii) a communication model which estimates the time for individual communication events, (iii) timing diagrams which abstractly represent both the communication pattern and the network performance, and (iv) scheduling algorithms which reduce overall communication time by appropriately positioning the communication events in the timing diagram.

Our approach is a general one, and can be used for different collective communication patterns and a variety of network-based architectures. In Section 4, we use our scheduling framework for the all-to-all personalized communication pattern in a typical metacomputing environment. We develop three different scheduling algorithms for this problem. Our first algorithm is a matching-based algorithm. It first constructs a bipartite graph, with edge weights equal to communication costs between processor pairs. A series of maximum weight complete matchings in this graph are then computed. The communication schedule is derived from these matchings. Our second algorithm is a greedy approximation to this matching-based algorithm. The third algorithm is a heuristic that has been used for the open shop scheduling problem. We evaluate the performance of our algorithms by simulation, and compare it with a well-known scheduling technique used in homogeneous scenarios. Our results show excellent improvements in performance.

This paper represents one of the early efforts to formalize research problems related to network-based computing. In Section 6, we discuss other fundamental research issues that are motivated by the need for network-aware applications. We consider enhancements to our communication model and techniques to reduce the complexity of the scheduling algorithm. We also discuss communication scheduling in the presence of QoS constraints.

The rest of the paper is organized as follows: Section 2 discusses related research projects in metacomputing. Section 3 introduces our approach for deriving efficient communication techniques, and describes each of the components in detail. Section 4 formulates the all-to-all heterogeneous data communication problem, and presents our scheduling algorithms for this communication pattern. Section 5 presents simulation results of our algorithms. Section 6 discusses future research directions for network-aware communication scheduling. Section 7 concludes the paper.

2. Related Work

Several research projects are developing software infrastructure and defining API functionality for network-based computing systems. We believe that our work will complement these software development efforts. Our scheduling techniques from Section 4 can be incorporated into these software systems and tool-kits. A few projects are also investigating performance related issues.

The Globus project [9, 10] at ANL and USC-ISI is developing a set of low level core services, called the Globus tool-kit. This includes modules for resource location and allocation, communications, authentication, process creation, and data access. Higher level systems software and applications then build upon the functionality provided by the tool-kit. Nexus is the communications library component of the Globus tool-kit. Globus incorporates a directory service, called the Metacomputing Directory Service (MDS). Applications can query MDS for information on current loads on the processing nodes, as well as end-to-end network performance between node pairs.

The Legion project at the University of Virginia uses an object oriented approach to metacomputing system design [12, 18]. The philosophy is to hide the complexity of resource scheduling, load balancing, etc. from the application developer. The object oriented properties of encapsulation and inheritance, as well as software reuse, fault containment, and reduction in complexity are used to achieve this goal.

The Virtual Distributed Computing Environment (VDCE) at Syracuse University [25, 26] aims to develop a complete framework for application development, configuration, and execution. A GUI allows library routines or user developed routines to be combined into an application task graph. The task graph is then interpreted and configured to execute on currently available resources.

At Carnegie Mellon, the ReMoS (Resource Monitoring System) project [7] is developing a portable and system-independent API that allows applications to obtain information about network status and capabilities. Most architectures generate information about the network hardware and software in a system-specific format. ReMoS provides a standard interface format that is independent of the details of any particular type of network. ReMoS explicitly accounts for resource sharing between applications.

The Management System for Heterogeneous Networks (MSHN) [21] project at Naval Postgraduate School, USC, and Purdue University is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. Processes may have different priorities, deadlines, and compute characteristics. The goal is to assign resources to individual applications so that their QoS requirements are satisfied. MSHN also addresses uncertainty due to unpredictable loads in the operating environment. Various task mapping and scheduling algorithms are being developed [1, 20]. Our research is a part of the MSHN effort.

Communication performance in the presence of multiple heterogeneous networks has been investigated in [14, 15]. Experiments are performed on a local cluster of workstations, interconnected with ATM, Ethernet, and Fibre-Channel networks. The performance characteristics of each of the networks are first evaluated by measuring the time for sending messages of various sizes over the particular network. These characteristics are used to choose a suitable technique for data communication. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks. However, this research only considered point-to-point communication between a pair of nodes in the system. Collective communication patterns such as all-to-all or all-to-some were not studied. Such collective communication patterns typically occur in most parallel applications.

Distributed heterogeneous computing has important military applications as well. The BADD (Battlefield Awareness and Data Dissemination) [6] program at DARPA aims to develop an operational distributed data communication system. The goal is to deliver to warfighters an accurate, timely, and consistent picture of the battlefield, as well as to provide access to key transmission mechanisms and world-wide data repositories. [24] considers an important data staging problem that arises in such heterogeneous networking environments, where data items must be moved from their initial locations to requester nodes. Each data request also has a time-deadline and priority associated with it. In [24], a heuristic based on the multiple-source shortest-path algorithm is used to find a communication schedule for this data staging problem. In Section 6, we mention some related problems.

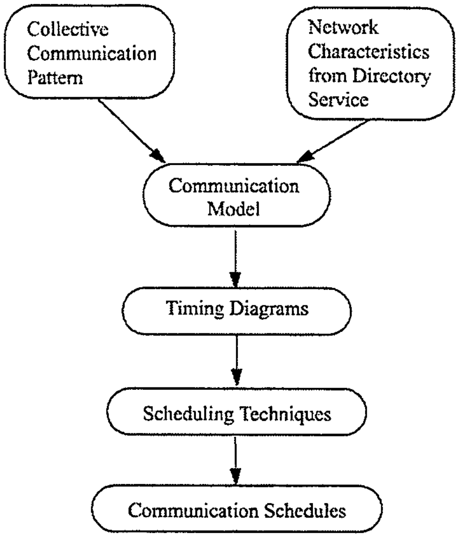

Collective Communication Pattern
Network Characteristics from Directory Service
Communication Model
Timing Diagrams
Scheduling Techniques
Communication Schedules

Figure 2. Our communication scheduling approach.

3. Our Approach: A Uniform Framework for Communication Scheduling

Figure 2 shows our approach for developing adaptive communication techniques, which are essential for network-aware applications. We use a communication scheduling framework consisting of four key components: (i) A directory service, (ii) An analytical communication model, (iii) Timing diagrams, and (iv) Scheduling algorithms. The directory service provides information on current network performance. Based on this information and the application's communication pattern, the communication model is used to compute the time for each node-to-node communication event. This is then represented using a timing diagram. A scheduling algorithm uses a timing diagram as input, and appropriately schedules the events to reduce the overall communication time. We discuss each of these components in further detail.

3.1. Directory Service

Since network load in shared environments varies with time, a directory service which provides information on current network performance is essential. A suitable directory infrastructure is therefore a key component of our framework for developing adaptive communication techniques.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		34.5	89.5	12	42
ANL	34.5		20	26.5	4.5
IND	89.5	20		42.5	21.5
USC-ISI	12	26.5	42.5		29.5
NCSA	42	4.5	21.5	29.5	

Table 1. Latency (ms) between 5 GUSTO sites.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		512	246	2044	391
ANL	512		491	693	2402
IND	246	491		311	448
USC-ISI	2044	693	311		4976
NCSA	391	2402	448	4976	

Table 2. Bandwidth (kbits/s) between 5 GUSTO sites.

The information provided by the directory makes it possible to develop communication schedules which are adaptive to changes in network performance. At run-time, applications can query the directory service through an Application Programming Interface. For example, the Metacomputing Directory Service (MDS) in Globus [8] provides current information on start-up costs and end-to-end bandwidths between every pair of processors. The ReMoS API, developed at CMU [7], is an example of an API that is independent of the details of network hardware.

Table 1 and 2 are examples of information provided by the directory service in GUSTO, which is a testbed of Globus. The directory provides current values of end-to-end network latency and bandwidth between any pair of computing sites. The tables show five of the GUSTO sites: NASA AMES, Argonne National Lab, University of Indiana, USC-ISI, and NCSA.

The directory service takes into account the current network load, including the load imposed by the application. If the paths between two distinct node pairs share a common link, the bandwidth of the common link is divided among these communicating pairs.

3.2. Communication Model

We use a communication model to analytically represent the network performance. Using information about the application's communication pattern and the performance parameters provided by the directory service, the communication model can estimate the time for individual node-to-node communication events.

Consider a typical metacomputing system, such as shown in Figure 1. A path between compute nodes typically includes links from multiple networks of different bandwidths. For example, in Figure 1, a message from a node in Site 1 to a node in Site 2 would pass through the local network at both sites and the long haul link which interconnects these geographically distributed sites.

Our communication model represents the network performance between any processor pair (P_i, P_j) using two parameters: a start-up cost T_{ij} and a data transmission rate B_{ij}. The time for sending a m byte message between these nodes is then given by $T_{ij} + \frac{m}{B_{ij}}$. The two parameters abstractly represent the total time for traversing all the links on the path between P_i and P_j. The model ignores the negligible delays incurred by contention at intermediate links and nodes on the path between P_i and P_j.

Our model focuses on the effective network performance at the application layer. We assume the availability of end-to-end send and receive communication routines, which can be invoked between any pair of processor nodes. Since the details of network topology, routing, and flow control policies are not visible to the application, our model does not incorporate these parameters.

A similar communication model has been widely used for tightly-coupled distributed memory systems with good results [27]. In metacomputing systems, typical values for the start-up cost could be in the range of 10 to 50 ms, while typical values for the bandwidth could be in the range of kb/s to hundreds of Mb/s.

The model assumes that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple messages to be transmitted simultaneously. Software support for non-blocking and multithreaded communication sometimes allow applications to initiate multiple send and receive operations. However, all these operations are eventually serialized by the single hardware port to the network. Our model accurately represents this phenomenon.

If multiple nodes simultaneously send to any node P_j, we say that node contention occurs at P_j. The model assumes that these messages are received one after the other at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation.

3.3. Timing Diagrams

We use timing diagrams to represent communication schedules for given network characteristics and a communication pattern. Examples of timing diagrams for all-to-all personalized communication with 5 processors are shown in Figures 4 and 7. The diagram consists of P columns, one per processor. The vertical axis represents time. The communication events in column i represent the messages sent from processor P_i. The rectangle labeled j in column i represents the message sent from P_i to P_j [1]. The height of the rectangle denotes the time for the communication event [2]. Once the message sizes and the values of T_{ij} and B_{ij} between all processor pairs are known, the heights of all the rectangles can be determined. Thus, the timing diagram inherently absorbs the heterogeneity in network parameters and message lengths.

3.4. Scheduling Algorithms

Our communication scheduling algorithms determine the positions of the individual events in the timing diagram so that the completion time is minimized. A valid communication schedule must satisfy the following conditions – since a node cannot send multiple message simultaneously, none of the rectangles in a column can overlap in time. Similarly, since multiple simultaneous receive events are not permitted at a processor, all the rectangles with the same label j must have mutually disjoint time intervals.

We do not consider "indirect" schedules where messages from different sources are combined at intermediate nodes and then forwarded to common destinations. This is because such combine-and-forward schemes increase the volume of traffic to be communicated. Since data in metacomputing applications is often extremely voluminous, this can lead to large communication costs.

We also do not allow messages to be partitioned. Since the start-up overhead is incurred for each message transmission, such a partitioning would increase the start-up overheads.

The next section presents our scheduling algorithms for the all-to-all personalized communication pattern.

4. Scheduling Algorithms for Total Exchange

In this section, we develop communication scheduling algorithms for total exchange, or all-to-all personalized communication. We briefly describe a well known communication algorithm for this problem. Section 5 shows the performance improvements achieved by our new algorithms over this algorithm.

4.1. Communication Pattern and Scheduling Complexity

All-to-all personalized communication occurs very frequently in HPC applications. For example, consider a two-dimensional matrix which is initially distributed by rows among the processors. If the matrix must be transposed so that the final distribution has columns on each processor, the resulting communication pattern is an all-to-all personalized communication. Here, each compute node has a distinct message for every other node in the system. For a P processor system, this communication pattern consists of $O(P^2)$ communication events. The message sizes between all pairs of nodes are not necessarily the same. When the network is heterogeneous, the individual communication events in the timing diagram will have different lengths. These communication events in the timing diagram must be efficiently scheduled. The goal is to reduce the *completion time* t_{max} of the communication schedule, *i.e.* the time at which the last communication event is completed. Observe that the completion time of the schedule cannot be less than the summation of send times or receive times at any processor, whichever is larger. This is therefore a *lower bound* t_{lb} on the completion time. To analyze the complexity of this communication scheduling problem, we first state it as a decision problem.

TOT_EXCH: Given a distributed heterogeneous system with P processors $(P_0, ..., P_{P-1})$, a deadline τ, and a $P \times P$ communication matrix $\mathbf{C}$, where $\mathbf{C}_{i,j}$ is the time for the communication event from P_j to P_i, $0 \leq i, j < P$ is there a communication schedule with completion time less than or equal to τ?

Theorem 1: *TOT_EXCH is NP-Complete for $P > 2$.*

Proof: The theorem can be proved by transformation from the open shop scheduling problem. The problem [5, 11] consists of m machines and n jobs. Each machine i performs task $t_{j,i}$ of job j. The execution time of all tasks are given in an $n \times m$ matrix. There are no dependences among the tasks of a job. Hence there are no restrictions on the sequence in which these tasks are to be executed. However, any machine can work on only one job at a time and any job can be processed by only one machine at a time. The goal is to schedule the tasks on the machines so as to minimize the finish time. The problem is known to be *NP*-Complete for $m > 2$ [11]. Details of our proof can be found in [3]. □

4.2. Baseline Algorithm

Since the total exchange communication scheduling problem is *NP*-Complete, we have developed heuristic algo-

[1] A receive schedule can be similarly constructed, where the communication events in column i represent messages received by processor P_i.

[2] The width of the rectangle does not have any significance.

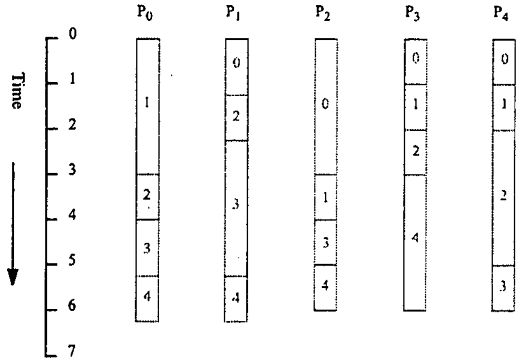


Figure 3. Example problem.

rithms. As a *baseline algorithm* for performance comparison with our heuristic algorithms, we shall use the caterpillar algorithm, which is widely used in tightly coupled homogeneous systems. This generates a schedule with P steps. In step j, $(0 \leq j < P)$, each compute node $P_i (0 \leq i < P)$ sends a message to $P_{(i+j) mod P}$ [13]. Such a schedule does not incur any node contention in a homogeneous system, when the message sizes and network bandwidths are uniform. This is because all the communication events have the same duration, *i.e.* all the rectangles in the timing diagram have the same height. An important disadvantage of the baseline algorithm is that it derives a fixed schedule, which is not adaptive to variations in message lengths or network performance.

We illustrate our scheduling techniques with a running example. Figure 3 shows an example communication problem, represented in the timing diagram formalism. The unscheduled communication events originating from each processor are shown in increasing order of destination processor number.

In our examples, we assume that the diagonal entries in **C** are zeroes. This is valid, since the time for a local memory copy operation is negligible in comparison with the time for sending messages over the heterogeneous network.

Using the baseline algorithm, the schedule shown in Figure 4 is derived. Observe that the longer communication events in the earlier steps cause the later communication steps to be delayed.

Theorem 2: Performance Bound for the Baseline Algorithm.

1. *The completion time t_{max} of the baseline schedule is always within $\frac{P}{2}$ times the lower bound t_{lb}.*
2. *The above bound is tight, i.e. there exist instances where the baseline schedule takes $\frac{P}{2}$ times the lower bound.*

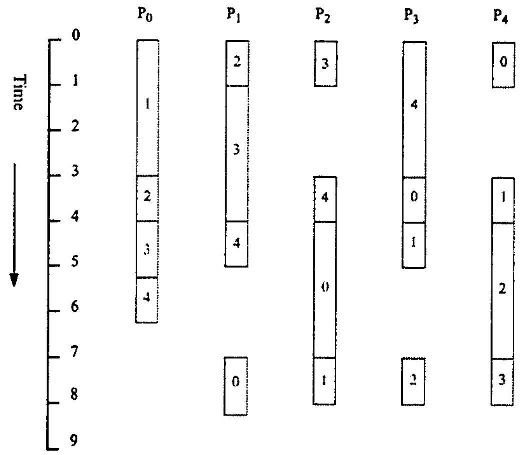


Figure 4. Schedule generated by baseline algorithm.

Proof: We first introduce the notion of a dependence graph **DG** for a given schedule. This is a directed graph with P^2 nodes, one for each communication event. A directed edge is present from node i to node j if there exists a sequential dependency between the corresponding communication events in the schedule.

Figure 5 shows the dependence graph for the baseline schedule with 5 processors. Column i contains all the communication events sent from Processor P_i, in the order that they appear in the schedule. Observe that the edges in **DG** are of two kinds: (i) vertical edges between adjacent nodes in the same column, and (ii) diagonal edges between nodes in adjacent columns. A correspondence exists between the graph **DG** and the communication matrix **C**. Each node in **DG** corresponds to an entry in **C**. If a vertical edge exists between two nodes, then these correspond to entries in the same column of **C**. If a diagonal edge is present, then the nodes correspond to entries in the same row of **C**.

Each path in the **DG** for the baseline schedule contains P nodes and $P - 1$ edges. The completion time is equal to the weight of the longest path in the graph. Let $t_1, t_2, \ldots, t_P$ be the nodes in the longest path. Then,

$$t_{max} = t_1 + t_2 + \ldots + t_P \qquad (1)$$

Since adjacent nodes in any path belong to the same row or column of **C**, and from the definition of t_{lb},

$$t_{lb} \geq max\{(t_1 + t_2), (t_3 + t_4), \ldots, (t_{P-1} + t_P)\} \qquad (2)$$

We can now rewrite Eq (1) as

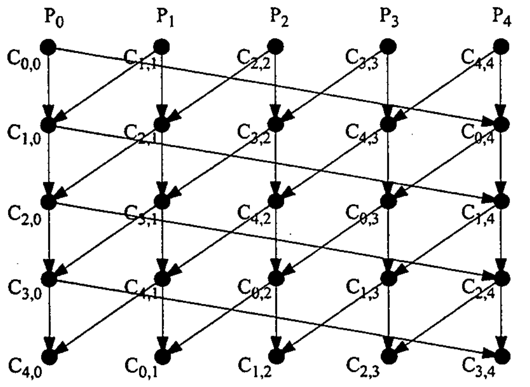


Figure 5. Dependence graph for the baseline schedule.

$$
\begin{aligned}
t_{max} &= (t_1 + t_2) + (t_3 + t_4) + \ldots + (t_{P-1} + t_P) \\
t_{max} &\leq \frac{P}{2} \times max\{(t_1 + t_2), \ldots, (t_{P-1} + t_P)\} \qquad (3)
\end{aligned}
$$

From Eq (2) and Eq (3)

$$t_{max} \leq \frac{P}{2} \times t_{lb} \qquad (4)$$

To prove the tightness of the bound, consider the following communication matrix:

$$C = \begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & 1 & \epsilon & \epsilon \\ 1 & 1 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon & \epsilon \end{bmatrix}$$

Each dependence path consists of 4 elements. The first element is always on the diagonal. Adjacent elements in the path are either in the same row or the same column. In the former case, the i^{th} element in the path is to the immediate left of the $i - 1^{th}$ element. In the latter case, the i^{th} element is immediately below the $i - 1^{th}$ element. For this example, the critical path contains all the unit-time entries, and takes 4 units of time. The lower bound is $2 + 2\epsilon$ [3].

Therefore,

$$\frac{t_{max}}{t_{lb}} = \frac{4}{2 + 2\epsilon} \approx 2 \qquad (5)$$

□

4.3. Matching-Based Scheduling Techniques

We present two matching-based scheduling techniques for the total exchange problem. The first technique finds a series of maximum weight matchings in a bipartite graph. We also consider the variation wherein minimum matchings are found. .

Our algorithm partitions the $P \times P$ communication events into P independent steps using graph matching algorithms. For a P node system, we construct a bipartite graph with P vertices on each side. The edge from v_i on the left side to v_j on the right side is assigned a weight equal to the time for the communication event from P_i to P_j. Thus, there are $O(P^2)$ edges in the bipartite graph. A complete matching in such a graph consists of P edges, and corresponds to a permutation of $(P_0, \ldots, P_{P-1})$. Such a matching can therefore represent a valid communication step, without contention at any processor. Well known algorithms exist for finding a maximum weight complete matching [17]. This is identical to the linear assignment problem. The complexity of this algorithm is $O(P^3)$. Our algorithm therefore consists of finding a maximum weight complete matching in the graph, deleting the edges of the matching from the graph, and then repeating the process until P such matchings have been found. Thus, the total complexity is $O(P^4)$. Although the schedule finds the communication events step by step, the communication phase does not impose a synchronization among the processors after each step. A communication event will begin whenever the sending and receiving processors are both ready.

In theory, the completion time of the matching based techniques can be $\frac{P}{2}$ times the lower bound. We can prove a result similar to part (i) of Theorem 2. In practice, the performance is significantly better, and the bound is therefore not tight. Unlike the fixed schedule derived by the baseline algorithm, our matching based schedule is adaptive. When the lengths of the communication events change with variations in network performance, the algorithm finds a different schedule with a low completion time. Section 5 presents simulation results.

For the example of Figure 3, our adaptive maximum matching algorithm derives the schedule shown in Figure 6. The matching technique groups together communication events with similar length, thereby reducing the idle cycles. Figure 6 is an optimal schedule for this example, since there exists a processor (P_1 or P_2) which is busy during the entire schedule.

4.4. Greedy Technique

The greedy technique is an approximation to the matching technique, with a lower computational complexity. Initially, the communication events within each processor are

[3] ϵ is an arbitrarily small numer.

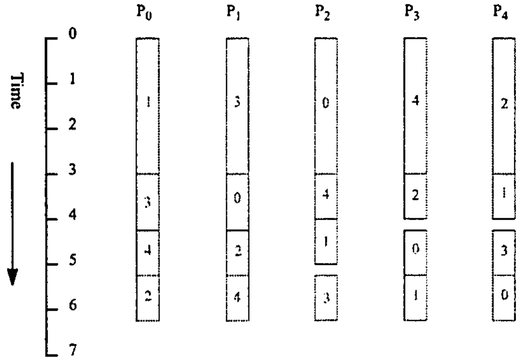


Figure 6. Schedule generated by a series of maximum matchings.

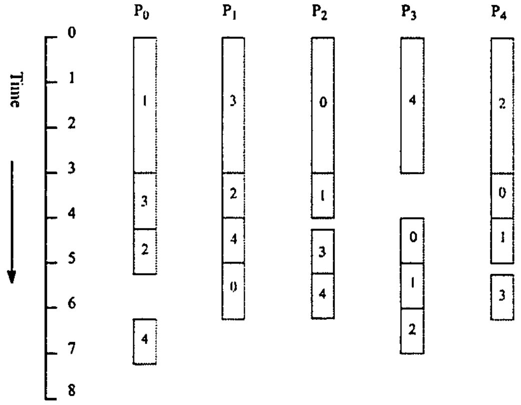


Figure 7. Schedule generated by the greedy algorithm.

rank ordered in decreasing order of communication time. A series of communication steps is then composed. In composing each step, we traverse the rank ordered list of every processor, with the goal of finding a destination processor. If a destination processor is found, it will be the first processor in its list that has not been selected by this processor in a previous step, and that is not the destination of another processor in the same step. If the end of the list is reached without finding a destination, the processor idles during this step, and we proceed to the next processor. Due to such incomplete steps, the total number of steps could be larger than P. To ensure fairness, a processor which was idle in any step will be the first to pick the destination processor in the next step. If there was no idle processor in a step, the last processor in any step will be the first in the next step. The greedy algorithm has a computational complexity of $O(P^3)$. From the description above, it is clear that the greedy algorithm is adaptive to the lengths of the communication events. For the previous example, the communication schedule derived by the greedy algorithm is shown in Figure 7.

4.5. Open Shop Technique

Since our communication problem has similarities to the open shop scheduling problem, we have developed a scheduling algorithm based on a heuristic derived for the open shop problem [22]. Other approximate algorithms for the open shop problem are given in [4].

Each processor is considered as two independent entities, a sender and a receiver. The following data structures are maintained by the algorithm:

- For each sender $i, 0 \leq i < P$, a set R_i of receivers is maintained. Initially, this consists of all receivers to which i must send a message. When a communication event is scheduled, the appropriate receiver is deleted from the receiver set.

- The P-element arrays *sendavail* and *recvavail* contain information about the availability of the corresponding senders and receivers. For example, the i^{th} element of *sendavail* specifies the earliest time at which sender i can participate in future send operations. All elements of both these arrays are initialized to 0.

The algorithm proceeds as follows:

- Whenever a sender i becomes available at time *sendavail[i]*, its receiver set R_i is scanned, and the earliest available receiver j is selected. The communication event from i to j is scheduled to begin at time *t=max(sendavail[i], recvavail[j])*. *sendavail[i]* and *recvavail[j]* are assigned the value $t + \mathrm{C}[j, i]$, since the sender i and receiver j will be busy until this time. Further, j is deleted from R_i.

- If multiple senders become available at the same time (for example, at time 0), they are processed in an arbitrary order. However, all senders that become available at time t are processed before any senders that become available at a later time. The algorithm maintains a list of senders in increasing order of their time of availability.

- Whenever a sender is finished with all its operations, it is deleted from this list. The algorithm terminates when all the senders are thus deleted.

The total number of communication events to be scheduled is $O(P^2)$. The scheduling of each event takes $O(P)$

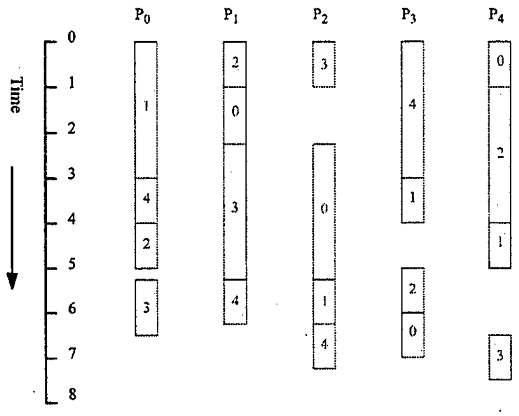


Figure 8. Schedule generated by the open shop algorithm.

time, since the elements of the corresponding receiver set must be scanned. The algorithm therefore runs in time $O(P^3)$.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to any of the elements in its receiver set. Idle cycles are inserted in a sender's schedule only if none of its potential receivers are available. For our running example, the schedule derived by this heuristic is shown in Figure 8.

Theorem 3: *The open shop heuristic algorithm is guaranteed to find a communication schedule whose completion time is within twice the lower bound.*

Proof: Assume, without loss of generality, that the last sender to finish all its transmissions is i. Let j be the receiver that i sends its last message to. It can be deduced that during the idle cycles in sender i's schedule, receiver j must have been always busy. If this were not the case, the algorithm would have scheduled the communication event from i to j at this time. Thus, we can conclude that the sum of the idle cycles in sender i's schedule is bounded by the total time for communication events having j as the receiver, *i.e.,* the sum of elements in row j of **C**. The completion time is the sum of the total time for send events from sender i, *i.e.,* the sum of elements in column i of **C**, and the idle cycles in sender i. Thus, the completion time is at most the sum of a row and a column in the communication matrix C, and is hence within twice the lower bound. □

5. Experimental Results

We have developed a software simulator that executes the scheduling algorithms discussed in Section 4, and calculates the completion time for each of them. The simulator accepts processor count and communication times as input, and generates the schedules based on these techniques. We have used this simulation tool to evaluate the baseline, maximum matching, minimum matching, greedy, and open shop scheduling techniques.

The simulator generates random performance characteristics for pairwise network performance, using information from the GUSTO directory service as a guideline. The communication matrix **C** can then be generated for any fixed message size. We have selected message sizes of 1kB, 1MB, and a random mix of these two sizes. In our experiments, we assume that the diagonal entries in **C** are zeroes. This is valid, since the time for a local memory copy operation is negligible in comparison with the time for sending messages over the heterogeneous network. The scheduling techniques are then applied to this communication matrix. Results for the different message sizes are shown in Figures 9, 10, and 11. Systems with up to 50 processors were considered.

Figure 12 considers a scenario when some of the processors are designated as servers. The message sizes from the servers to the other (client) processors are assumed to be large. The message sizes between the servers themselves and also between the client processors are small. This is typical in multimedia applications, where images and video clips reside on servers, and are accessed by other processors. In our experiment, 20% of the processors are assumed to be servers. Data is also assumed to be partitioned over the servers, so that the load on the servers is balanced. It can be seen that the baseline algorithm performs very poorly in such scenarios. Our algorithms perform 2 to 5 times faster than the baseline in these examples.

The graphs clearly show the performance improvements that can be achieved by our communication scheduling techniques. The open shop algorithm finds schedules that are very close to the lower bound, often within 2%, and always within 10 %. The maximum and minimum matching based techniques find schedules with comparable completion times. These are within 15% of the lower bound. The schedules generated by the greedy algorithm are within 25% of the lower bound. The schedules generated by the baseline algorithm sometimes take upto 6 times longer than the lower bound. Based on our results, the open shop algorithm achieves the best performance.

6. Enhancements and Future Research

In the previous sections, we presented our approach for developing communication scheduling techniques that are adaptive to network performance variations. To the best of our knowledge, this is one of the early efforts in formalizing communication problems relevant to network-based computing. Several exciting research issues remain to be explored. In this section, we discuss some future research di-

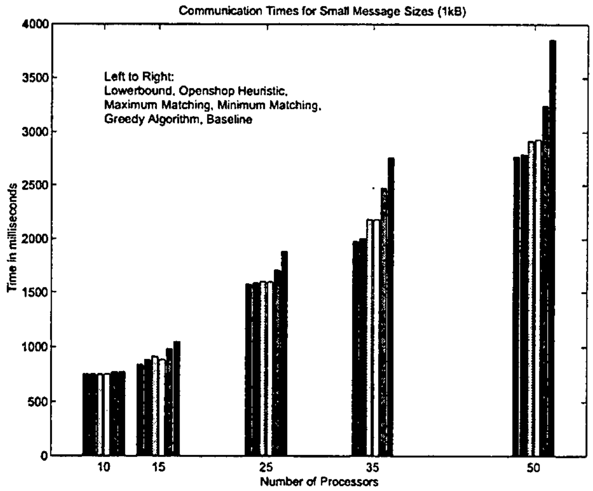


Figure 9. Simulator results for all-to-all personalized communication with small message sizes.

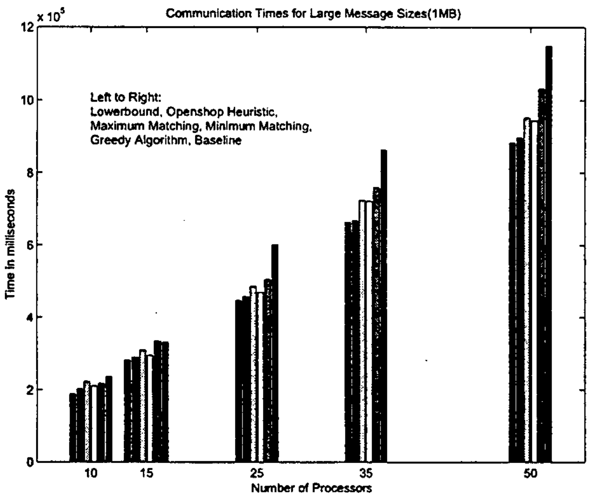


Figure 10. Simulator results for all-to-all personalized communication with large message sizes.

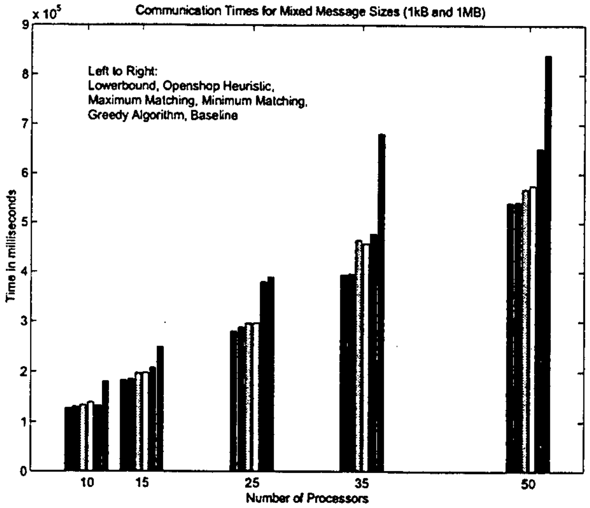


Figure 11. Simulator results for all-to-all personalized communication with mixed message sizes.

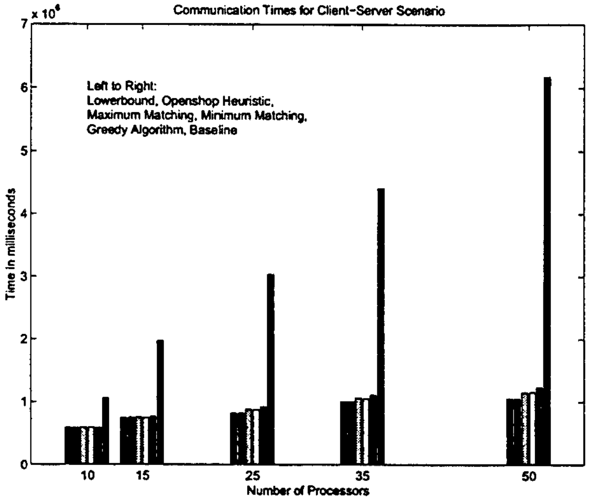


Figure 12. Simulator results for all-to-all personalized communication when 20% of the processors are servers. Servers send large messages to their clients.

rections that are motivated by the need for network-aware communication scheduling.

6.1. Enhancing the Model

Our scheduling algorithms were developed using a simple yet effective communication model. In Section 3, we mentioned the assumptions made by our model, and the validity of these assumptions. Enhanced versions of the model can be formulated by relaxing some of these assumptions. For example, one restriction is that a processor can send and receive only one message at a time. This restriction can be relaxed in two ways.

When multiple messages arrive at a node, we can assume that the messages are received in an interleaved fashion. For example, the use of multithreading allows multiple simultaneous communication events in Nexus. An additional parameter α can be introduced for the overhead incurred in context switching between the multiple receiving threads. Thus, if t_1 and t_2 are the times for individually receiving two messages, the total time for receiving them simultaneously would be $(1+\alpha)(t_1+t_2)$.

It could also be assumed that a finite buffer space is available at nodes to receive messages. When multiple messages arrive at a node, one of the messages is received by the application, while the others are queued in the buffer. The sending nodes do not wait until the receive operation is complete, but only until the message is stored in the buffer. If the buffer is full, the sender must wait until adequate free space is created in the buffer.

6.2. Incremental Dynamic Scheduling

The communication schedules presented in Section 4 are computed at run-time based on information obtained from the directory service. In many sensor-based applications, a series of continuously arriving data sets are processed in an identical manner. In such cases, the overhead for repeatedly calculating the communication schedule at run-time can be expensive, especially when the number of processors is large. It is therefore necessary to develop scheduling techniques which have significantly lower computational costs.

An incremental approach would be one way to reduce the complexity of deriving dynamic communication schedules. Here, a communication schedule is computed once, either at compile time or during the first run-time occurrence. At each subsequent invocation, the incremental algorithm must refine this communication schedule to find a new communication schedule. The algorithm would query the directory service regarding changes in the bandwidths. In this context, the research problem is that of developing fast algorithms for refining an existing communication schedule.

6.3. Enhancing Adaptivity of the Schedules

In some scenarios, the lengths of all communication events may not be known even when the communication is started. This could happen because variations in network performance are so rapid that significant changes could occur within the duration of the communication schedule. In such cases, an initial communication schedule can be derived using estimates of the communication times. The schedule can then be modified at intermediate *checkpoints*. At these checkpoints, processors decide whether the difference between the estimated time and actual time is large enough to require rescheduling. The checkpoints could be defined in different ways: after each communication event is complete ($O(P)$ checkpoints), or after half the remaining communication events are complete ($O(\log P)$ checkpoints), and so on.

6.4. Scheduling with Critical Resources or QoS Constraints

We have discussed communication schedules where the goal is to minimize the completion time. In many scenarios, other cost measures are also important. For example, one of the processors in the heterogeneous system could be a critical resource (*e.g.,* an expensive supercomputer). The schedule should complete the communication events of this processor as early as possible, even if it delays the other processors.

Quality of Service (QoS) requirements in some applications can introduce other variations in the problem formulation. For example, data forwarding and data staging problems arise in the BADD project [6]. The QoS parameters associated with each message are deadlines and priorities. The communication schedule must ensure that data items reach their destinations by the specified real-time deadlines. When multiple communication events contend for a communication link, the scheduling algorithm must sequence them based on their respective deadlines and priorities.

7. Conclusion

In this paper, we have developed a uniform framework for communication scheduling in heterogeneous network-based systems. The framework consists of a directory service, a communication model, timing diagrams, and scheduling algorithms. We discussed our approach for the design of adaptive communication techniques, and applied it to the problem of all-to-all personalized communication. Although this problem has been thoroughly researched for homogeneous systems, we showed that well known algorithms perform poorly in the presence of network heterogeneity. We developed algorithms based on bipartite graph matching, and a heuristic algorithms based on Open shop scheduling. We showed that our algorithms performed significantly better than a well known homogeneous communication scheduling algorithm. Our algorithms are adaptive and execute at run-time, based on network performance information obtained from the directory service. To the best of our knowledge, this is one of the early efforts in formalizing communication problems in a distributed heterogeneous computing environment. Our paper also discusses several new research problems related to communication scheduling in network-based systems, that arise due to the unique features of such environments. These include communication scheduling with QoS constraints and techniques to reduce the complexity of the scheduling algorithm.

Acknowledgments

We thank Roy Jonker of MagicLogic Optimization, Inc. for his public domain program to solve the Linear Assignment Problem. This routine was used to find the matchings in our algorithms. We also thank Craig Lee and Paul Stelling of The Aerospace Corporation and the members of the MSHN project for helpful technical suggestions.

References

[1] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in runtime predictions. In *Proc. Heterogeneous Computing Workshop*, pages 79–87, March 1998.

[2] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *IEEE Trans. Parallel and Distributed Systems*, 6(2):154–164, February 1995.

[3] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. *Manuscript*, Dept. of EE-Systems, University of Southern California, May 1998.

[4] H. Brasel, T. Tautenhahn, and F. Werner. Constructive heuristic algorithms for the open shop problem. *Computing*, 51:95–110, 1993.

[5] P. Brucker. *Scheduling Algorithms*. Springer, 1995.

[6] DARPA ISO Web Page for the BADD Program. *https:// www.iso.darpa.mil/ WD@27000.cgi?get+iso:: Office+Information_System +WDI_i_home_frames*.

[7] T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland. ReMoS: A resource monitoring system for network-aware applications. Technical Report CMU-CS-97-194, School of Computer Science, Carnegie Mellon University, Dec 1997.

[8] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewskiand, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Intnl. Symp. on High Performance Distributed Computing*, pages 365–375, 1997.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[10] Globus Web Page. *http://www.globus.org*.

[11] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.

[12] A. S. Grimshaw and W. A. Wulf. Legion – a view from 50,000 feet. In *Proc. Fifth IEEE Intl. Symp. on High Performance Distributed Computing*, August 1996.

[13] L. H. Jamieson, P. T. Mueller, and H. J. Siegel. FFT algorithms for SIMD parallel processing systems. *Journal of Parallel and Distributed Computing*, 3(1):48–71, March 1986.

[14] J. Kim and D. J. Lilja. Exploiting multiple heterogeneous networks to reduce communication costs in parallel programs. In *Proc. Heterogeneous Computing Workshop*, pages 83–95, April 1997.

[15] J. Kim and D. J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intl. Symp. High Performance Distributed Computing*, 1997.

[16] H. Korab and M. D. Brown, *Eds. Virtual Environments and Distributed Computing at SC '95: GII Testbed and HPC Challenge Applications on the I-WAY*. ACM/IEEE Supercomputing '95, 1995.

[17] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[18] Legion Web Page. *http://legion.virginia.edu*.

[19] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient algorithms for block-cyclic redistribution of arrays. *Algorithmica*, to appear.

[20] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Proc. Heterogeneous Computing Workshop*, pages 57–69, March 1998.

[21] MSHN Web Page. *http://www.cs.nps.navy.mil/mshn*.

[22] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, June 1994.

[23] L. Smarr and C. E. Catlett. Metacomputing. *Commns. of the ACM*, 35(6):45–52, June 1992.

[24] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, March 1998.

[25] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, I. Ra, D. Kim, Y. Kim, X. Bing, and B. Ye. The software architecture of a virtual distributed computing environment. In *Proc. Sixth IEEE Intl. Symp. on High Performance Distributed Computing*, 1997.

[26] VDCE Web Page. *http://www.atm.syr.edu/projects/vm/index.html*.

[27] C.-L. Wang, P. B. Bhat, and V. K. Prasanna. High-performance computing for vision. *Proceedings of the IEEE*, 84(7):931–946, July 1996.

Appears in the Proceedings of the 11th International Conference on Parallel and Distributed Computing Systems (PDCS 1998)

Block-Cyclic Redistribution over Heterogeneous Networks

Prashanth B. Bhat [*] and Viktor K. Prasanna[*][†]
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
{prabhat + prasanna}@halcyon.usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
raghu@aero.org

Abstract

Clusters of workstations and networked parallel computing systems are emerging as promising computational platforms for HPC applications. The processors in such systems are typically interconnected by a collection of heterogeneous networks such as Ethernet, ATM, and FDDI, among others. In this paper, we develop techniques to perform block-cyclic redistribution over P processors interconnected by such a collection of heterogeneous networks.

We represent the communication scheduling problem using a timing diagram formalism. Here, each interprocessor communication event is represented by a rectangle whose height denotes the time to perform this event over the heterogeneous network. The communication scheduling problem is then one of appropriately positioning the rectangles so as to minimize the completion time of all the communication events. For the important case where the block size changes by a factor of K, we develop a heuristic algorithm whose completion time is atmost twice the optimal. The running time of the heuristic is $O(PK^2)$.

Our heuristic algorithm is adaptive to variations in network performance, and derives schedules at run-time, based on current information about available network bandwidth. Our experimental results show that our schedules always have communication times that are very close to optimal.

Keywords: *Workstation clusters, heterogeneous networks, communication scheduling, block-cyclic redistribution.*

1. Introduction

Due to advances in high-speed networks, workstation clusters and loosely connected distributed systems are being used as platforms for High Performance Computing. Wide area networking technology has also enabled the development of *metacomputers* [11], wherein grand challenge applications are parallelized across geographically distributed supercomputers and visualization devices. Such distributed systems are typically interconnected with a collection of many different kinds of communication networks, such as ATM, HiPPI, and Ethernet.

Prototype systems with such heterogeneous networks have been built. For example, [6] evaluated the performance of HPC applications on a cluster of workstations interconnected with ATM and FDDI networks. The I-WAY (Information Wide Area Year) metacomputer at SC '95 consisted of over 10 networks of varying bandwidths, protocols, and routing technology. The HiPer-D project investigates the use of networked distributed computing capabilities in battle management systems on U.S. Navy cruisers. The Battlefield Awareness and Data Dissemination (BADD) program develops techniques for delivering multimedia data to mobile troops over a combination of wired and wireless networks [12].

From the above examples, it is clear that heterogeneity is a salient characteristic of the interconnection network in most distributed computational environments. Further, the network is shared among multiple applications. The performance therefore depends upon the current traffic conditions, and typically varies over time.

For scalable performance on such a platform, support for fast application-level communication is necessary. Efficient implementations of important collective communication kernels must be incorporated into communication libraries. In this paper, we develop communication techniques for *block-cyclic redistribution* over such heterogeneous networks. We consider the important case where the block size changes by a factor of K. Our techniques can also be extended to other redistribution problems.

The block-cyclic distribution is widely used in many HPC applications to partition an array over multiple processors. For example, in signal processing applications, the block-cyclic distribution is the natural choice for radar and sonar data cubes. Many of the frequently occurring communication patterns, such as the corner turn operation, can be then viewed as block-cyclic redistribution opera-

[*] Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract numbcr N62271-97-M-0931.

[†] Partially supported by NSF undcr grant CCR-9317301.

tions. ScaLAPACK, a widely used mathematical software for dense linear algebra computations, also uses a block-cyclic distribution for good load balance and computational efficiency. Matrix transpose operations, which often occur in linear algebra computations, are a special case of the block-cyclic redistribution. HPF provides directives for specifying block-cyclic distribution and redistribution of arrays.

The problem of block-cyclic redistribution in a tightly-coupled homogeneous parallel system has been well researched. However, the heterogeneity and sharing of the network make it necessary to develop new communication scheduling techniques. In Section 4, we present a communication scheduling algorithm that is well suited for heterogeneous networks. The algorithm is adaptive to variations in network performance. The schedule is derived at run-time, based on current information about network load.

Our scheduling approach is based on a communication model that represents the communication performance between every processor pair using two parameters: a start-up time and a data transmission rate. We formalize the communication scheduling problem using a timing diagram representation. Each interprocessor communication event is represented as a rectangle whose height equals the time to perform the communication over the heterogeneous network. The height is calculated using our communication model. The communication scheduling problem is then one of appropriately positioning the rectangles in the timing diagram so as to minimize the completion time of all the communication events. Our heuristic algorithm derives a communication schedule whose completion time is always within twice the optimal. The running time of the heuristic is $O(PK^2)$, where P is the number of processors, and K is the factor by which the block size changes.

The rest of the paper is organized as follows. Section 2 discusses the characteristics of the block-cyclic redistribution communication pattern. Section 3 discusses some previous research efforts on block cyclic redistribution. Section 4 introduces our communication model for the heterogeneous network and presents our heuristic algorithm for block-cyclic redistribution. Section 5 presents performance results from the experimental implementation of our algorithm. Section 6 concludes the paper and discusses future research directions.

2. The Block-Cyclic Redistribution Problem

The block-cyclic distribution of an array can be defined as follows [14]: given P processors, an array with N elements, and a block size x, the distribution first partitions the array elements into contiguous blocks of x items each. b_i is the i^{th} block, $0 \leq i < \frac{N}{x}$[1]. The blocks are then assigned to processors in a round robin fashion so that b_i is assigned to processor (i mod P). We denote a block-cyclic distribution of block size x as *cyclic*(x).

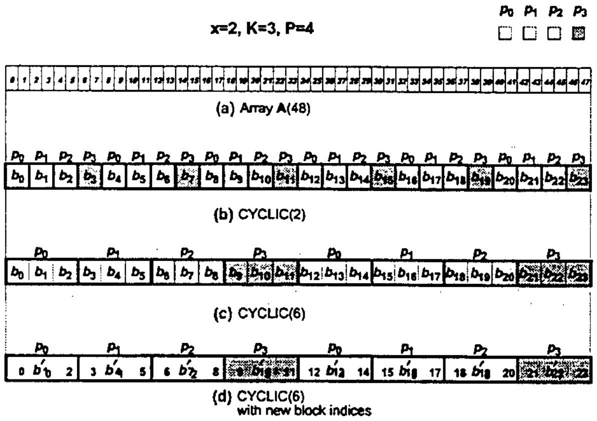


Figure 1. Redistribution from *cyclic*(2) to *cyclic*(6) on 4 processors.

The block-cyclic data redistribution problem consists of reorganizing an array from one block-cyclic distribution to another. The most frequently encountered version of this redistribution problem is the *cyclic*(x) to *cyclic*(Kx) redistribution, which is the problem we consider in this paper. We denote the *cyclic*(x) to *cyclic*(Kx) redistribution among P processors as $\Re_x(K, P)$.

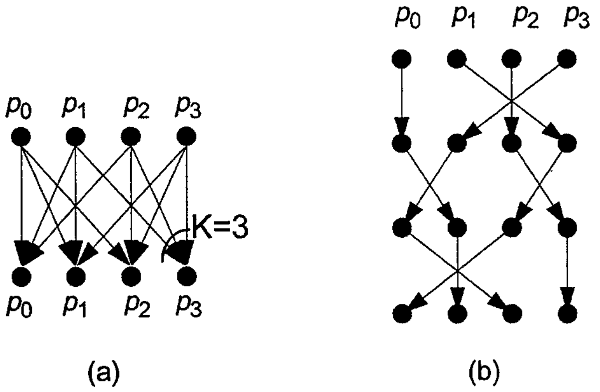
Figure 1 shows the example of $\Re_2(3, 4)$. The array **A** which has $N = 48$ elements is shown in Figure 1(a). Figure 1(b) shows the initial distribution, *cyclic*(2). Here, b_i is of size 2 elements, and has a global block index i. The blocks are assigned to $P(= 4)$ processors in a round robin fashion. If the block size is increased by a factor of $K(= 3)$, *i.e.*, the new block size becomes 6, each set of three consecutive blocks becomes a new block, as shown in Figure 1(c) and (d).

Block-cyclic redistribution consists of three main phases:

1. **Index set computation and message generation:** Each processor computes indices of array elements that are to be communicated with the other processors, as well as the destination processors of such array elements. The elements are then packed into message buffers, one for each destination processor.

2. **Communication scheduling:** A given processor contains messages for a total of K processors, and will also receive messages from K processors. The aim of communication scheduling is to reduce the overall communication time. During this phase, each processor determines an ordering among its send and receive events,

[1] For simplicity, we assume that x divides N.



(a)

(b)

Figure 2. (a) Communication pattern for $\Re_x(3,4)$ (b) Contention-free schedule for a homogeneous network.

so as to reduce contention.

3. **Interprocessor communication:** The processors send and receive messages in the order specified by the communication schedule. This phase incurs software start-up overheads for invocation of the send and receive system calls, and transmission costs for sending data over the interconnection network. In the absence of communication scheduling, this phase can become very inefficient due to node contention.

The interprocessor communication pattern of $\Re_x(K,P)$ can be represented by a communication graph, shown in Figure 2(a). Each edge represents a message that is to be sent between the corresponding processors. Note that each processor P_i, $0 \leq i < P$ must send messages to K destinations, and receive messages from K sources. For given values of x, K, and P, the position of edges in this graph and the array indices corresponding to each edge are computed during the index computation phase. In [7], we have developed efficient techniques for index computation, for systems with homogeneous networks.

Figure 2(b) shows an example of a K-step communication schedule for $\Re_x(3,4)$. Here, the communication pattern of Figure 2(a) is broken down into a series of contention-free communication steps. Node contention occurs when multiple processors simultaneously send messages to a receiver. When the network is homogeneous, all the communication events within any step of Figure 2(b) would take the same amount of time. In [7], we have developed communication scheduling techniques for such a homogeneous scenario. However, when the network links are heterogeneous, the time taken for each message varies with the available network bandwidth between the corresponding processors. Due to this non-uniformity in message communication times, node contention and idle cycles would be introduced in the schedule of Figure 2(b) if it was used without modification. It is therefore necessary to develop new communication scheduling techniques for cyclic redistribution over heterogeneous networks. Section 4 presents our new algorithms for this problem.

3. Related Work

The block-cyclic redistribution problem has been the focus of several research efforts. Techniques have been developed for both the index computation phase and the communication scheduling phase over a homogeneous network. In [13], Choudhary *et. al.* present efficient index computation algorithms for $\Re_x(K,P)$, when $P \bmod K = 0$. They also consider the redistribution from *cyclic*(x) to *cyclic*(y), for general x and y, using *gcd* and *lcm* methods.

In [9], Banerjee *et. al.* represent a *cyclic*(x) distribution as a set of strided line segments. Using this formalism, the array elements to be exchanged between a pair of processors is computed by the intersection of the respective line segments.

Sadayappan *et. al.* [5] and Walker *et. al.* [14] have developed algorithms for the communication scheduling phase. Here, a K step schedule is given for $\Re_x(K,P)$. At each step, processors exchange data in a contention-free manner: each processor sends data to exactly one processor and receives data from exactly one processor.

In [7], we introduced a uniform framework to develop redistribution algorithms for $\Re_x(K,P)$. Based on this framework, efficient algorithms were developed for reducing both the index computation and communication overheads. Three classes of techniques were presented for $\Re_x(K,P)$: direct, indirect, and hybrid. In the direct approach, a block is sent directly from a source processor to its destination without being sent to intermediate processors. The direct approach performs the $\Re_x(K,P)$ communication in K communication steps. The indirect approach performs $\Re_x(K,P)$ in atmost $\lceil \log_2 K \rceil + 2$ steps. Here, the array elements are communicated in a "combine and forward" manner. The hybrid approach is a combination of the direct and indirect approaches.

In [2], communication schedules are developed for the general redistribution problem of *cyclic*(r) over a set of P processors, to *cyclic*(s), over a different set of Q processors. Graph matching algorithms are used to develop communication schedules in this work. These techniques have two important drawbacks: (i) The communication scheduling phase is expensive, due to the use of graph matching algorithms, and (ii) All processors are synchronized after each step in the interprocessor communication phase. This increases the interprocessor communication time.

[8] considers the problem of run-time redistribution from *cyclic*(x) on P processors to *cyclic*(Kx) on Q processors, over a homogeneous network. The algorithm is based on a generalized circulant matrix formalism. The generated schedule minimizes the number of communication steps and eliminates node contention in each communication step. The network bandwidth is fully utilized by ensuring that equal-sized messages are transferred in each communication step.

Performance studies of heterogeneous networks were reported in [6]. Experiments were performed on a local cluster of workstations, interconnected with ATM, Ethernet, and Fibre-Channel networks. The performance characteristics of each of the networks were first evaluated by sending messages of various sizes over the particular network. These characteristics were used to choose a suitable technique for communication over the heterogeneous network. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks. However, this research only considered point-to-point communication between a pair of nodes in the system. In comparison, our paper investigates the collective communication pattern of block-cyclic redistribution.

The Management System for Heterogeneous Networks (MSHN) project at Naval Postgraduate School, USC, and Purdue University is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. The goal is to schedule processor and network resources among individual applications so that their QoS requirements are satisfied. In this context, data staging techniques for distributed systems with heterogeneous networks have been considered [12].

4. Our Communication Scheduling Approach

As discussed in Section 2, block-cylic redistribution consists of index computation, communication scheduling, and interprocessor communication. Since the index computation phase is independent of the network characteristics, techniques developed for homogeneous networks [7] can be used. In this section, we consider the communication scheduling phase. We first discuss the assumptions and communication model that we shall use to analyze our communication schedule. Section 4.3 presents our communication scheduling algorithm.

4.1. Communication Model

The overall network in the P processor system consists of several heterogeneous network components. Each component interconnects a subset of the processors. We can model such a network as a completely connected virtual network with heterogeneous performance between each pair of processors. Thus, the path between any pair of processors can be modeled as a single link, with the effective performance of the heterogeneous path. Techniques to aggregate the performance of different networks into a single virtual network have been considered, and are an active area of research [6]. We model the communication performance of the path between a pair of processors P_i and P_j by a start-up cost $T_{i,j}$ and a data transmission rate $B_{i,j}$. Thus, to send a m byte message between P_i and P_j, the time taken is $T_{i,j} + \frac{m}{B_{i,j}}$. When the message sizes are large, the data transmission cost is the dominating component, and the start-up cost can be ignored [2]. Typical values for the start-up cost could be in the range of 10 to 50 μ s, while typical values for the bandwidth could be in the range of a few Mb/s to hundreds of Mb/s.

We assume that the effective network performance between any pair of processors will not change during the communication phase. This can be ensured if the application reserves network bandwidth for the duration of the communication. [3] and [15] discuss issues relating to reserving network resources.

We assume that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple distinct messages to be transmitted simultaneously. If multiple nodes simultaneously send to any node P_j, these messages are received one after the other at P_j. We say that node contention occurs at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation. We do not consider the use of wild-card non-blocking receives. Although this can allow a processor to simultaneously wait for many receives, large buffer space overheads are incurred.

4.2. Timing Diagrams

Communication schedules can be conveniently represented by timing diagrams. An example of a timing diagram for $\Re_x(3,4)$ is shown in Figure 3. A timing diagram consists of P columns, one per processor. The vertical axis rep-

[2] We make this approximation in our experiments.

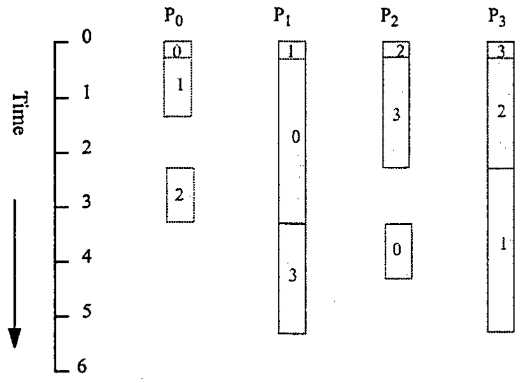


Figure 3. A communication schedule for $\Re_x(3,4)$.

resents time. The communication events in column i represent the messages sent from processor P_i. The rectangle labeled j in column i represents the message sent from P_i to P_j [3]. The height of the rectangle denotes the time for the communication event. The width of the rectangle does not have any significance. Once the message sizes and T_i and B_i parameters for all processors are known, the heights of all the rectangles can be determined. Thus, the timing diagram inherently absorbs the heterogeneity in network parameters and message lengths.

Any communication scheduling algorithm must determine the positions of the communication events so that an efficient schedule is found. The goal is to find the schedule that has the minimum *completion time, i.e.,* the time at which the last communication event is completed. Since a node cannot send multiple messages simultaneously, no overlap is allowed among any of the rectangles in a column. Similarly, since multiple simultaneous receive events are not permitted at a processor, all the rectangles with the same label j must have mutually disjoint time intervals. Thus, the completion time of the schedule cannot be less than the summation of send times or receive times at any processor, whichever is larger. This quantity is therefore a *lower bound* on the completion time of any schedule.

4.3. Our Scheduling Algorithm

During the index computation and communication scheduling phases, each processor independently computes the entire schedule. The communication matrix C, which represents the time for each point-to-point communication event, is computed based on the values of P, K, N, and the network performance parameters. C$[i, j]$ is the height of the rectangle labeled i in column j of the timing diagram.

It can be shown that the problem of finding the optimal communication schedule is NP-complete. We have therefore developed a heuristic algorithm for this problem. Each processor is considered as two independent entities, a sender and a receiver. The following data structures are maintained by the algorithm:

- For each sender $i, 0 \leq i < P$, a set R_i of receivers is maintained. These are the receivers to which i must send a message sometime during the schedule. For the $\Re_x(K,P)$ communication pattern, each set R_i will initially consist of K elements. The R_i's are obtained from the communication matrix in a straightforward way. These are the row indices of the non-zero elements in column i of C.

- The P-element arrays *sendavail* and *recvavail* contain information about the availability of the corresponding senders and receivers. For example, the i^{th} element of *sendavail* specifies the earliest time at which sender i can participate in future send operations. All elements of both these arrays are initialized to 0.

The algorithm proceeds as follows:

- Whenever a sender i becomes available at time *sendavail[i]*, its receiver set R_i is scanned, and the earliest available receiver j is selected. The communication event from i to j is scheduled to begin at time *t=max(sendavail[i], recvavail[j]). sendavail[i]* and *recvavail[j]* are assigned the value $t + \mathrm{C}[j, i]$, since the sender i and receiver j will be busy until this time. Further, j is deleted from R_i.

- If multiple senders become available at the same time (for example, at time 0), they are processed in an arbitrary order. However, all senders that become available at time t are processed before any senders that become available at a later time. The algorithm maintains a list of senders in increasing order of their time of availability.

- Whenever a sender is finished with all its operations, it is deleted from this list. The algorithm terminates when all the senders are thus deleted.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to one of the elements in its receiver set. Idle cycles are inserted in a sender's schedule only if none of its potential receivers are available. Our algorithm is also *adaptive* to changes in network performance. The derived communication schedule depends on the entries of C, which are in turn

[3] A receive schedule can be similarly constructed, where the communication events in column i represent messages received by processor P_i.

dependent on network load conditions. The schedule can be derived at runtime, using current values of network performance parameters. Previous redistribution algorithms for homogeneous networks do not provide such adaptivity. A "fixed" communication schedule is used irrespective of network load and bandwidth.

The total number of communication events to be scheduled is PK. The scheduling of each event takes $O(K)$ time, since the elements of the corresponding receiver set must be scanned. Our scheduling algorithm therefore runs in $O(PK^2)$ time. On a single node of the Cray T3E, our algorithm executed in about 10 ms for $P = 64$ and $K = 63$. This is the cost of the communication scheduling phase.

Based on the schedule, the processors perform send and receive operations during the interprocessor communication phase. Initially, a single non-blocking receive and a non-blocking send is posted by each processor. If the receive finishes first, the next receive operation is immediately posted. If the send finishes first, the next send operation is initiated. This continues until all the communication events have been completed. The cost of the interprocessor communication phase depends upon the completion time of the schedule.

Lemma: *The heuristic algorithm is guaranteed to find a communication schedule whose completion time is within twice the optimal.*

Proof: Assume, without loss of generality, that the last sender to finish all its transmissions is i. Let j be its last receiver in the schedule, *i.e.,* j is the receiver that i sends its last message to. It can be deduced that during the idle cycles in sender i's schedule, receiver j must have been always busy. If this were not the case, the greedy algorithm would have scheduled the communication event from i to j at this time. Thus, we can conclude that the sum of the idle cycles in sender i's schedule is bounded by the total communication time for events incident at receiver j, *i.e.,* the sum of elements in row j of **C**. The completion time is the sum of idle cycles in sender i and the total time for send events from sender i, *i.e.,* the sum of elements in column i of **C**. Thus, the completion time is at most the sum of a row and a column in the communication matrix **C**, and is hence within twice the lower bound. □

Our communication problem has some similarities to the open shop scheduling problem [4]. Here, a set of M machines execute a set of N jobs. Each job J_i has a task to be executed on every machine M_j, the execution time for which is given by $t_{i,j}$. The tasks may be executed in any order. However, atmost one machine may process a given job at any time. Also, atmost one job may be processed by a given machine at any time. The goal is to schedule the tasks on the machines to minimize the completion time. The problem is known to be NP-complete, and an algorithm similar to the above greedy heuristic has been used [10].

5. Experimental Results

In this section, we evaluate the performance of our heuristic scheduling technique by measuring the time for the interprocessor communication phase. We compare this with the time for the interprocessor communication phase of the direct communication schedule, which is a fixed communication schedule. For $\Re_x(K, P)$, this schedule breaks down the communication pattern into K contention-free steps. It has been shown to be effective in systems with homogeneous networks, when each communication event takes the same amount of time. It can be derived from the formalisms of [5], [7], and [14].

We have developed a simulator that accepts as input the network performance parameters, and the parameters of the cyclic redistribution problem. The simulator then derives the communication schedule, and calculates the expected completion time. We also implement the communication schedules on a Cray T3E, and measure the time taken to perform the redistribution. Our experimental methodology is summarized in the following key steps:

1. For an input value of P, our simulator first generates performance parameters for the heterogeneous network. The simulator accepts as input the minimum and maximum bandwidth values, and randomly generates bandwidth values in this range for every processor pair.

2. For given input values of K and N, the communication matrix **C** is then generated. The lower bound on any communication schedule is calculated as the maximum sum over all the rows and columns of **C**.

3. Our heuristic scheduling algorithm described in the previous section is then executed, and the sender and receiver schedules at each processor are computed. The simulator also computes the estimated completion time of the schedule.

4. We implement the schedule generated in Step 3, on a Cray T3E. Although the network of the Cray T3E is homogeneous, we can simulate the heterogeneous network of Step 1 by scaling the sizes of the messages appropriately. Thus, if the heterogeneous network has a bandwidth of B_1 between nodes i and j, and if B_2 is the node-to-node bandwidth of the Cray T3E, then we scale the message size between node i and j by a factor $\frac{B_2}{B_1}$. Each point-to-point event in the communication schedule is implemented by using MPI send and receive calls.

5. We also implement the direct schedule on the Cray T3E. The communication performance of our heuristic schedule is compared with that of the direct schedule, for various values of P and K. The communication times are measured and tabulated.

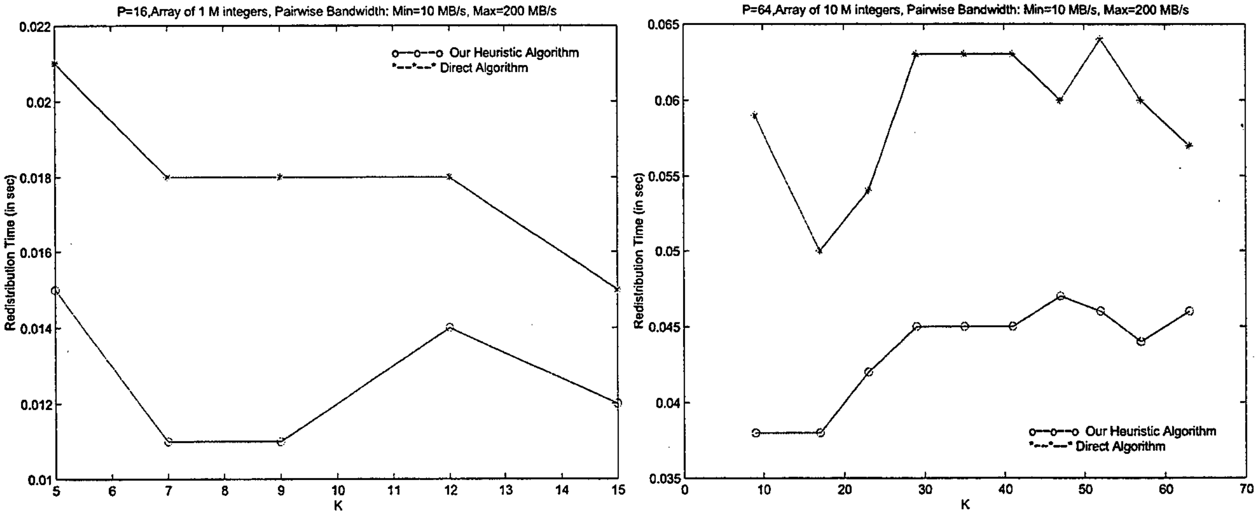


Figure 4. Interprocessor communication times of our heuristic schedule and the direct schedule on 16 and 64 processors of the Cray T3E.

Figure 4 compares the interprocessor communication time of our heuristic schedule and the direct schedule. These times were measured by our experimental implementations of both schedules on the Cray T3E. The simulated heterogeneous network had node-to-node bandwidths in the range of 10 MB/s to 200 MB/s. The Cray T3E has a bandwidth of 150 MB/s, which is the value we use for B_2 in Step 4 above. In Figure 4, results for 16 and 64 processors are shown. K is varied from a small number to $P - 1$. Experimental results for other values of P can be found in [1]. From Figure 4, we can conclude that our heuristic schedule achieves significant and consistent performance improvements over the direct schedule. The communication time of the heuristic schedule is lower than that of the direct schedule by 10% to 60%.

We observe that, for a fixed value of P, the communication time for both the schedules varies considerably with the value of K. This phenomenon is not observed in [7], where the direct schedule is implemented on a homogeneous network. The variation occurs due to the heterogeneity in the network. The communication matrix for $\Re_x(K, P)$ has PK non-zero entries, while other entries are zeroes. For different values of K, messages are exchanged between a different subset of the total P^2 processor pairs.

Figure 5(a) shows the estimated completion times for the heuristic and direct schedules, as calculated by the simulator. The lower bound on the completion time is also shown in these figures. It can be seen that the completion time of our heuristic algorithm is always within 2 - 10 % of the lower bound.

An important characteristic of our heuristic algorithm is that the schedule is adaptive to variations in network performance. In a metacomputing system, applications can query a directory service for current values of network performance. Our heuristic can use such information during the communication scheduling phase. In contrast, the direct algorithm uses a fixed communication pattern, irrespective of the network performance. Figure 5(b) shows the simulated communication time of both schedules as the heterogeneity in the network is varied. Here, $P = 64$, $K = 40$, and $N = 10000000$. The variation in network bandwidth is increased from 100 to 1900%. When the variation is 100 %, the network bandwidth varies in the range 10 MB/s to 20 MB/s. When the variation is 1900 %, the network bandwidth varies in the range of 10 MB/s to 200 MB/s. The completion time of our heuristic algorithm is always very close to the lower bound, for all values of network heterogeneity.

6. Conclusion

In this paper, we have developed an effective communication scheduling technique for the important problem of block-cyclic redistribution over a heterogeneous network. Our techniques target an emerging class of computational platforms, namely workstation clusters and distributed systems. The interconnection network in such systems is typically heterogeneous and shared. Our adaptive algorithms derive communication schedules using run-time information on network performance and heterogeneity.

Our scheduling techniques can lead to significant improvements in application performance. Our experimental results show reductions of upto 60 % in communication

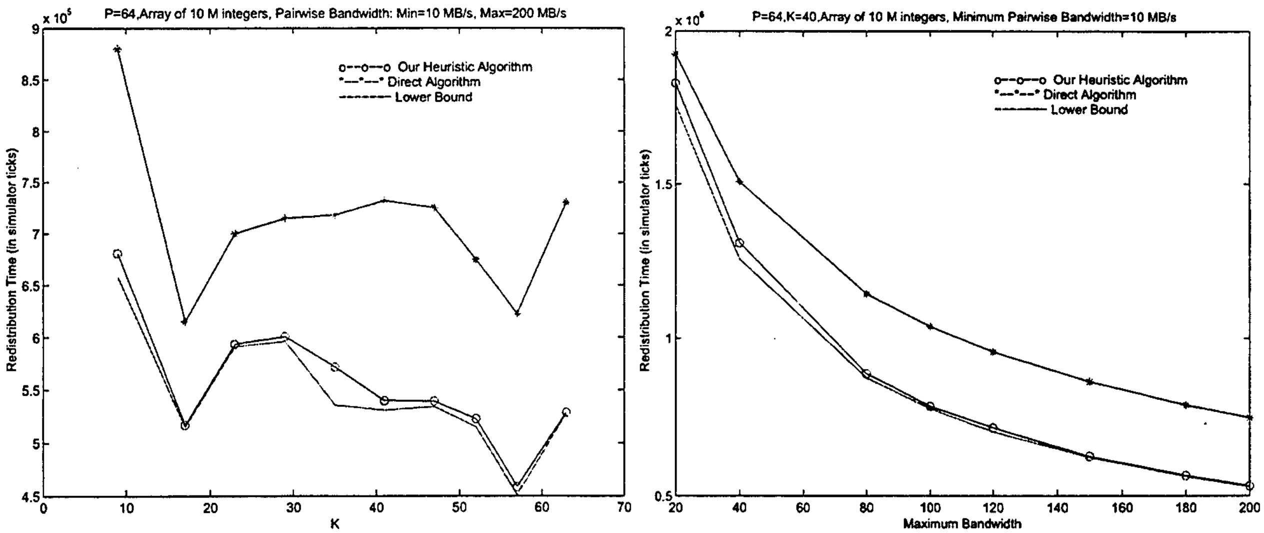


Figure 5. Simulation results: lower bound, heuristic schedule, and direct algorithm. (a)P=64 processors, K=9 to 63 (b) P=64, K=40, network heterogeneity varies from 100% to 1900%.

time, compared with widely used algorithms for homogeneous systems.

In our future research efforts, we shall study the performance of our algorithm using a real heterogeneous network. We are also developing techniques to reduce the cost of the communication scheduling phase, by using enhanced data structures. Our communication scheduling technique can be easily extended to other important redistribution problems, such as block-cyclic redistribution of multi-dimensional arrays, and redistribution from *cyclic*(x) to *cyclic*(y).

References

[1] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Block-cyclic redistribution over heterogeneous networks. *Manuscript*, Dept. of EE-Systems, University of Southern California, July 1998.

[2] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling block-cyclic array redistribution. *IEEE Trans. Parallel and Distributed Systems*, 9(2):192–205, Feb. 1998.

[3] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. In *Proc. 5th Intnl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.

[4] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.

[5] S. D. Kaushik, C. H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase array redistribution: Modeling and evaluation. In *Proc. Intnl. Parallel Processing Symposium*, pages 441–445, Apr. 1995.

[6] J. Kim and D. J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intnl. Symp. High Performance Distributed Computing*, 1997.

[7] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient algorithms for block-cyclic redistribution of arrays. *Algorithmica*, to appear.

[8] N. Park, V. K. Prasanna, and C. S. Raghavendra. Efficient algorithms for block-cyclic array redistribution between processor sets. In *Proc. Supercomputing '98*.

[9] S. Ramaswamy and P. Banerjee. Automatic generation of efficient array redistribution routines for distributed memory multicomputers. In *Proc. 5th Symposium on Frontiers of Massively Parallel Computation*, pages 342–349, Feb. 1995.

[10] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Computing*, 23(3):617–632, June 1994.

[11] L. Smarr and C. E. Catlett. Metacomputing. *Commns. of the ACM*, 35(6):45–52, June 1992.

[12] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, Mar. 1998.

[13] R. Thakur, A. Choudhary, and J. Ramanujam. Efficient algorithms for array redistribution. *IEEE Trans. Parallel and Distributed Systems*, 7(6):587–594, June 1996.

[14] D. W. Walker and S. W. Otto. Redistribution of block-cyclic data distributions using MPI. Technical Report ORNL/TM-12999, Oak Ridge National Labs, June 1995.

[15] L. C. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Wittig. Issues of reserving resources in advance. In *Proc. 5th Intnl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.

Adaptive Communication Algorithms for Distributed Heterogeneous Systems *

Prashanth B. Bhat [1] and Viktor K. Prasanna [1]
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
{prabhat, prasanna}@halcyon.usc.edu

and

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
raghu@aero.org

*A preliminary version of this paper appears in the Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC 1998).

[1] Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

Abstract

Many grand challenge applications can benefit from metacomputing, *i.e.* the coordinated use of geographically distributed heterogeneous supercomputers. A salient feature of such systems is the heterogeneity in the network performance between different processor pairs. This paper considers the problem of performing efficient application-level communication in such heterogeneous network-based systems. We present a uniform communication scheduling framework, for developing adaptive communication schedules for various collective communication patterns. The framework enables schedules to be developed at run-time, based on network performance information obtained from a directory service. Based on this framework, we have developed communication schedules for the total exchange communication pattern. Our first algorithm develops a schedule by computing a series of matchings in a bi-partite graph. We also present a heuristic algorithm, based on the open shop scheduling problem. The completion time of the heuristic is guaranteed to be within twice the optimal. Simulation results show performance improvements of a factor of 5 over well known homogeneous scheduling techniques. This paper is one of the early efforts in formalizing and solving communication problems for metacomputing systems. We discuss several research issues that must be addressed to enable efficient collective communication in such environments.

Keywords: Collective communication, open shop scheduling, heterogeneous networks, metacomputing, adaptive communication scheduling, graph matching.

Running Head: Communication Algorithms for Heterogeneous Systems

Contact Author:

Viktor K. Prasanna

Department of EE-Systems, EEB 200C

University of Southern California

Los Angeles, CA 90089-2562

Email: prasanna@halcyon.usc.edu

Phone: (213)740-4483, Fax: (213)740-4418

1 Introduction

With recent advances in high-speed networks, *metacomputing* has emerged as a viable and attractive computational paradigm. A metacomputing system [23] consists of geographically distributed supercomputers and visualization devices. These are interconnected by a heterogeneous collection of local and wide-area networks. High performance applications can be executed over such a *networked virtual supercomputer,* wherein the distributed computational resources are used in a coordinated way, very much as though they were part of a single computer system.

The potential of metacomputing has been demonstrated by the Global Information Infrastructure (GII) testbed at SC '95 [17]. The testbed linked dozens of high performance computers and visualization machines with existing high-bandwidth networks and telephone systems. Many computationally intensive scientific applications were demonstrated using the testbed. Some of the leading metacomputing research projects are MSHN, Globus, Legion, and VDCE, to name a few. These will be discussed further in Section 2.

Figure 1 shows an example of a small-scale metacomputing system. The compute nodes in the system are located at three different sites. Some of the nodes are high-end supercomputing systems, while others are workstations. The example shown in this figure has three kinds of interconnection networks: (i) the multi-stage interconnection network within the IBM SP-2, (ii) local networks at each site, and (iii) high bandwidth long haul ATM or T3 links between the sites. The compute nodes at a site share the bandwidth of the long haul link when communicating with nodes at a remote site.

Although network-based computing platforms offer significant advantages for high performance computing, effective use of their resources is still a major challenge. Computational and communication resources are typically shared among different applications. Computational tasks may be preempted by higher priority processes. Network conditions change continuously, and run-time loads cannot be determined apriori. Applications must therefore be capable of *adapting* to changing system conditions.

In this paper, we develop communication techniques that enable applications to adapt to variations in network conditions. We focus on collective communication patterns among application processes executing over a heterogeneous network. Our goal is to develop efficient application-level implementations of these communication routines. We assume the availability of end-to-end send and receive communication routines, which can be invoked between any pair of nodes. The details of network topology, routing, and flow control policies are therefore hidden from the application.

Efficient algorithms for various collective communication patterns have been developed for tightly coupled parallel architectures with homogeneous networks [1, 20]. These algorithms have been incorporated into communication libraries and implementations of MPI. However, these techniques can perform poorly in metacomputing systems due to the heterogeneity among network bandwidths. Further, these are static algorithms, with no provision for adaptivity to network conditions.

In Section 3, we introduce our approach for developing network-aware communication techniques. The key components of our approach are: (i) a directory service which provides information on current network performance, (ii) a communication model which estimates the time for individual communication events, (iii) abstract formalisms which represent both the communication pattern and the network performance, and (iv) scheduling algorithms which reduce overall communication time by appropriately positioning the communication events in the abstract formalism.

Our approach is a general one, and can be used for different collective communication patterns and a variety of network-based architectures. In Section 4, we use our scheduling framework for the all-to-all personalized communication pattern in a typical metacomputing environment. We develop three different scheduling algorithms for this problem. Our first algorithm is a matching-based algorithm. It first constructs a bi-partite graph, with edge weights equal to communication costs between processor pairs. A series of maximum weight complete matchings in this graph are then computed. The communication schedule is derived from these matchings. Our second algorithm is a greedy approximation to this matching-based algorithm. The third algorithm is a heuristic that has been used for the open shop scheduling problem. We evaluate the performance of our algorithms by simulation, and compare it with a well-known scheduling technique used in homogeneous scenarios. Our results show excellent improvements in performance.

This research represents one of the early efforts to formalize research problems related to network-based computing [2]. In Section 6, we discuss other fundamental research issues that are motivated by the need for network-aware applications. We consider enhancements to our communication model and techniques to reduce the complexity of the scheduling algorithm. We also discuss communication scheduling in the presence of QoS constraints.

The rest of the paper is organized as follows: Section 2 discusses related research projects in metacomputing. Section 3 introduces our approach for deriving efficient communication techniques, and describes each of the components in detail. Section 4 formulates the all-to-all heterogeneous data communication problem, and presents our scheduling algorithms for this communication pattern. Section 5 presents simulation results of our algorithms. Section 6 discusses future research directions for network-aware communication scheduling. Section 7 concludes the paper.

2 Related Work

Several research projects are developing software infrastructure and defining API functionality for network-based computing systems. We believe that our work will complement these software development efforts. Our scheduling techniques from Section 4 can be incorporated into these software systems and tool-kits. A few projects are also investigating performance related issues.

The Management System for Heterogeneous Networks (MSHN) [21, 14] project is a collaborative effort between DoD (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). Our research is a part of the MSHN effort. MSHN (pronounced "mission") is designing and implementing a Resource Management System (RMS) for metacomputing systems. The main goal of MSHN is to determine the best way to support the execution of many different applications, each with its own quality of service (QoS) requirements, in a distributed, heterogeneous environment. Typical metacomputing environments consist of a mix of real-time, interactive, I/O-intensive, network intensive, and compute intensive applications, each with its own QoS requirements, including security issues. Real-time applications must meet specified time deadlines, while network intensive applications must be provided with guaranteed bandwidth and network attributes on network channels. MSHN's RMS schedules and passively monitors distributed and heterogeneous resources in shared environments so as to deliver acceptable end-to-end QoS for a collection of applications.

The Globus project [10, 11] at ANL and USC-ISI is developing a set of low level core services, called the Globus tool-kit. This includes modules for resource location and allocation, communications, authentication, process creation, and data access. Higher level systems software and applications then build upon the functionality provided by the tool-kit. Nexus is the communications library component of the Globus tool-kit. Globus incorporates a directory service, called the Metacomputing Directory Service (MDS). Applications can query MDS for information on current loads on the processing nodes, as well as end-to-end network performance between node pairs.

The Legion project at the University of Virginia uses an object oriented approach to metacomputing system design [13, 19]. The philosophy is to hide the complexity of resource scheduling, load balancing, etc. from the application developer. The object oriented properties of encapsulation and inheritance, as well as software reuse, fault containment, and reduction in complexity are used to achieve this goal.

The Virtual Distributed Computing Environment (VDCE) at Syracuse University [25] aims to develop a complete framework for application development, configuration, and execution. A GUI allows library routines or user developed routines to be combined into an application task graph. The task graph is then interpreted and configured to execute on currently available resources.

At Carnegie Mellon, the ReMoS (Resource Monitoring System) project [8] is developing a portable and system-independent API that allows applications to obtain information about network status and capabilities. Most architectures generate information about the network hardware and software in a system-specific format. ReMoS provides a standard interface format that is independent of the details of any particular type of network. ReMoS explicitly accounts for resource sharing between applications.

Communication performance in the presence of multiple heterogeneous networks has been investigated in [15, 16]. Experiments are performed on a local cluster of workstations, interconnected with ATM, Ethernet, and Fibre-Channel networks. The performance characteristics of each of the networks are first evaluated by measuring the time for sending messages of various sizes over the particular network. These characteristics are used to choose a suitable technique for data communication. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks. The sizes of the individual parts depend inversely on the speed of the network over which the part is transmitted. However, this research only considered point-to-point communication between a pair of nodes in the system. Collective communication patterns such as all-to-all or all-to-some were not studied. Such collective communication patterns typically occur in most parallel applications.

Distributed heterogeneous computing has important military applications as well. The BADD (Battlefield Awareness and Data Dissemination) [7] program at DARPA aims to develop an operational distributed data communication system. The goal is to deliver to warfighters an accurate, timely, and consistent picture of the battlefield, as well as to provide access to key transmission mechanisms and worldwide data repositories. [24] considers an important data staging problem that arises in such heterogeneous networking environments, where data items must be moved from their initial locations to requester nodes. Each data request also has a time-deadline and priority associated with it. In [24], a heuristic based on the multiple-source shortest-path algorithm is used to find a communication schedule for this data staging problem. In Section 6, we mention some related problems.

3 Our Approach: A Uniform Framework for Communication Scheduling

Figure 2 shows our approach for developing adaptive communication techniques in a system with dynamically varying network performance. We use a communication scheduling framework consisting of four key components: (i) An analytical communication model, (ii) A directory service, (iii) Abstract formalisms, and (iv) Scheduling algorithms. The communication model is an analytical representation of the network's characteristics, as observed at the application level. At run-time, the directory service provides current values for the model's performance parameters. Based on this information and the knowledge of the communication pattern, the time for each node-to-node communication event is calculated using the communication model. The communication scheduling problem is then represented using an abstract formalism. A scheduling algorithm positions the communication events in this formalism to reduce the overall completion time. The result is the desired communication schedule. We discuss each of these components in further detail.

3.1 Communication Model

We use a communication model to analytically represent the network performance. Using information about the application's communication pattern and the performance parameters provided by the directory service, the communication model can estimate the time for individual node-to-node communication events.

Consider a typical metacomputing system, such as shown in Figure 1. A path between compute nodes typically includes links from multiple networks of different bandwidths. For example, in Figure 1, a message from a node in Site 1 to a node in Site 2 would pass through the local network at both sites and the long haul link which interconnects these geographically distributed sites.

Our communication model represents the network performance between any processor pair (P_i, P_j) using two parameters: a start-up cost T_{ij} and a data transmission rate B_{ij}. The time for sending a m byte message between these nodes is then given by

$$T_{ij} + \frac{m}{B_{ij}} \tag{1}$$

The two parameters abstractly represent the total time for traversing all the links on the path between P_i and P_j. The model ignores the negligible delays incurred by contention at intermediate links and nodes on the path between P_i and P_j.

A similar communication model has been widely used for tightly-coupled distributed memory systems with good results [26]. In metacomputing systems, typical values for the start-up cost could be in the range of 10 to 50 ms, while typical values for the bandwidth could be in the range of kb/s to hundreds of Mb/s.

Our model represents the effective network performance as seen at the application layer. Since the low-level details of network protocols and routing are not visible to the application, our model does not incorporate these details. We assume the availability of point-to-point send and receive communication routines, which can be invoked between any pair of processor nodes. Communication libraries in current metacomputing testbeds, such as the Nexus library of Globus, provide such point-to-point `send` and `receive` routines. Such a library provides the abstraction of an efficient end-to-end channel between each pair of processor nodes. The `send` and `receive` are implemented efficiently, taking into consideration the details of network topology, routing, flow control policies, and heterogeneous network protocols. These low-level details are hidden below the end-to-end `send` and `receive` calls. Techniques to efficiently perform point-to-point communication over a collection of heterogeneous networks have also been investigated [15, 16].

The model assumes that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple messages to be transmitted simultaneously. Software support for non-blocking and multithreaded communication sometimes allow applications to initiate multiple send and receive operations. However, all these operations are eventually serialized by the single hardware port to the network. Our model accurately represents this phenomenon.

If multiple nodes simultaneously send to any node P_j, we say that node contention occurs at P_j. The model assumes that these messages are received one after the other at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation.

3.2 Directory Service

Since network load in shared environments varies with time, a directory service which provides information on current network performance is essential. A suitable directory infrastructure is therefore a key component of our framework for developing adaptive communication techniques. The information provided by the directory makes it possible to develop communication schedules which are adaptive to changes in network performance. At run-time, applications can query the directory service through an Application Programming Interface. For example, the Metacomputing Directory Service (MDS) in Globus [9] provides current information on start-up costs and end-to-end bandwidths between every pair of processors. The ReMoS API, developed at CMU [8], is an example of an API that is independent of the details of network hardware.

Table 1 and 2 are examples of information provided by the directory service in GUSTO, which is a testbed of Globus. The directory provides current values of end-to-end network latency and bandwidth between any pair of computing sites. The tables show five of the GUSTO sites: NASA AMES, Argonne National Lab, University of Indiana, USC-ISI, and NCSA.

The directory service takes into account the current network load, including the load imposed by the application. If the paths between two distinct node pairs share a common link, the bandwidth of the common link is divided among these communicating pairs.

3.3 Abstract Formalism

We use an abstract formalism to represent the communication scheduling problem. The formalism is constructed using timing information calculated by the communication model. It incorporates information obtained from the directory service, as well as information about the communication pattern. Several abstract formalisms are possible. We discuss three formalisms: bi-partite weighted graphs, communication matrices, and timing diagrams.

A communication matrix C is a $P \times P$ matrix that represents the time for each pairwise communication event. C_{ij} is the communication time for the message from P_i to P_j. The entries in C are calculated using the communication model $T_{ij} + \frac{m}{B_{ij}}$. Run-time values of T_{ij} and B_{ij} are obtained from the directory service. Eq (2) shows the structure of a communication matrix for a system with four nodes.

$$\mathrm{C} = \begin{bmatrix} \mathrm{C}_{0,0} & \mathrm{C}_{0,1} & \mathrm{C}_{0,2} & \mathrm{C}_{0,3} \\ \mathrm{C}_{1,0} & \mathrm{C}_{1,1} & \mathrm{C}_{1,2} & \mathrm{C}_{1,3} \\ \mathrm{C}_{2,0} & \mathrm{C}_{2,1} & \mathrm{C}_{2,2} & \mathrm{C}_{2,3} \\ \mathrm{C}_{3,0} & \mathrm{C}_{3,1} & \mathrm{C}_{3,2} & \mathrm{C}_{3,3} \end{bmatrix} \tag{2}$$

In the bi-partite graph formalism, the P processors are represented as a bi-partite graph of $2P$ nodes. The P^2 edges in this graph are each assigned a weight equal to the communication time between the corresponding pair of processors. The edge from v_i on the left side to v_j on the right side is assigned a weight equal to the time for the communication event from P_i to P_j. Figure 3 shows an example of the bi-partite graph formalism for a system with four nodes. For clarity, only some of the edge weights are shown.

Examples of timing diagrams for all-to-all personalized communication with 5 processors are shown in Figures 6 and 9. The diagram consists of P columns, one per processor. The vertical axis represents time. The communication events in column i represent the messages sent from processor P_i. The rectangle labeled j in column i represents the message sent from P_i to P_j [2]. The height of the rectangle denotes the time for the communication event [3]. Once the message sizes and the values of T_{ij} and B_{ij} between all processor pairs are known, the heights of all the rectangles can be determined. Thus, the timing diagram inherently absorbs the heterogeneity in network parameters and message lengths.

3.4 Scheduling Algorithms

The events in the abstract formalism are scheduled using communication scheduling algorithms. We develop different classes of communication scheduling algorithms for the different abstract formalisms.

When a timing diagram formalism is used, our communication scheduling algorithms determine the positions of the individual rectangles in the timing diagram so that the completion time is minimized. A valid communication schedule must satisfy the following conditions – since a node cannot send multiple message simultaneously, none of the rectangles in a column can overlap in time. Similarly, since multiple simultaneous receive events are not permitted at a processor, all the rectangles with the same label j must have mutually disjoint time intervals.

When a bi-partite graph formalism is used, our algorithm finds a series of maximum matchings in the graph. These represent the steps in the communication schedule. In the

[2] A receive schedule can be similarly constructed, where the communication events in column i represent messages received by processor P_i.

[3] The width of the rectangle does not have any significance.

set-based formalism, nodes are moved from a receiver set to a sender set, in a manner similar to shortest-path algorithms in directed graphs. We shall discuss these techniques in greater detail in the following chapters.

We do not consider "indirect" schedules where messages from different sources are combined at intermediate nodes and then forwarded to common destinations. This is because such combine-and-forward schemes increase the volume of traffic to be communicated. Since data in metacomputing applications is often extremely voluminous, this can lead to large communication costs.

We also do not allow messages to be partitioned. Since the start-up overhead is incurred for each message transmission, such a partitioning would increase the start-up overheads.

The next section presents our scheduling algorithms for the all-to-all personalized communication pattern. We have also developed scheduling algorithms for cyclic redistribution [3], broadcast and multicast [4, 2].

4 Scheduling Algorithms for Total Exchange

In this section, we develop communication scheduling algorithms for total exchange, or all-to-all personalized communication. We briefly describe a well known communication algorithm for this problem. Section 5 shows the performance improvements achieved by our new algorithms over this algorithm.

4.1 Communication Pattern and Problem Complexity

All-to-all personalized communication occurs very frequently in HPC applications. For example, consider a two-dimensional matrix which is initially distributed by rows among the processors. If the matrix must be transposed so that the final distribution has columns on each processor, the resulting communication pattern is an all-to-all personalized communication. Matrix transpose is a well known example of such a communication pattern. Here, each compute node has a distinct message for every other node in the system. For a P processor system, this communication pattern consists of $O(P^2)$ communication events. The message sizes between all pairs of nodes are not necessarily the same. When the network is heterogeneous, the individual communication events in the timing diagram will have different lengths. These communication events in the timing diagram must be efficiently scheduled. The goal is to reduce the *completion time* t_{max} of the communication schedule, *i.e.* the time at which the last communication event is completed. To analyze the complexity of this communication scheduling problem, we first state it as a decision problem.

TOT_EXCH: Given a distributed heterogeneous system with P processors $(P_0, \ldots, P_{P-1})$, a deadline τ, and a $P \times P$ communication matrix $\mathbf{C}$, where $\mathbf{C}_{i,j}$ is the time for the communication event from P_j to P_i, $0 \leq i, j < P$ is there a communication schedule with completion time less than or equal to τ?

Theorem 1 *TOT_EXCH is NP-Complete for $P > 2$.*

Proof: It is easy to see that *TOT_EXCH* belongs to *NP*. A Turing machine can guess the optimal schedule and verify that the completion time is less than τ. The verification phase must ensure that for each communication event i in column j of the timing diagram, $(0 \leq i, j < P)$, (i) the finish time of the event is less than τ, and (ii) this event does not overlap with any other event in column j or other events labeled i in the timing diagram. This takes $O(P)$ time per communication event, since there are P events in column j and

P events labeled i in the timing diagram. Since there are $O(P^2)$ communication events, the complexity of the verification phase is $O(P^3)$.

To prove that *TOT_EXCH* is *NP*-Hard, we show that a known *NP*-Complete problem *open shop*, can be polynomially transformed to this problem.

The open shop problem [6, 12] consists of m machines and n jobs. Each machine i performs task $t_{j,i}$ of job j. The execution time of all tasks are given in an $n \times m$ matrix. There are no dependences among the tasks of a job. Hence there are no restrictions on the sequence in which these tasks are to be executed. However, any machine can work on only one job at a time and any job can be processed by only one machine at a time. The goal is to minimize the finish time. The problem is known to be *NP*-Complete for $m > 2$ [12].

Consider an instance of the open shop problem with $n = m$. From this, we can construct an instance of *TOT_EXCH* as follows: Let the number of processors P be equal to n (or m). Construct the communication matrix with each element equal to the corresponding entry of the task execution time matrix, *i.e.*, $\mathrm{C}_{i,j} = t_{i,j}$, $(0 \leq i, j < P)$. Now, the constraint that a machine can work on only one job at a time is equivalent to the constraint that there should be no overlap between two communication events in any given column of the timing diagram. The constraint that a job can be processed by only one machine at a time is equivalent to the constraint that there should be no overlap between any two communication events with the same label in the timing diagram. The equivalence between the two problems can be seen in Figure 4. For a given deadline τ, it is easy to see that a communication schedule with completion time less than or equal to τ exists if and only if there exists an open shop schedule whose finish time is less than or equal to τ. Since the open shop problem is known to be *NP*-Complete, *TOT_EXCH* is also *NP*-Complete. □

4.2 Lower Bound

Due to the NP-Completeness of the *TOT_EXCH* problem, it is not possible to find optimal communication schedules. In the subsequent sections, we shall therefore present heuristic schedules. To evaluate the quality of these heuristics, we first develop a lower bound on the completion time.

Recall that each row in the communication matrix **C** corresponds to a sender, and each column corresponds to a receiver. We know that any processor (say P_i) cannot send multiple messages at any given time. Thus, the P messages from P_i cannot be completed at a time earlier than the sum of the entries in row i of **C**.

$$t_{max} \geq \sum_{j=0}^{(P-1)} \mathbf{C}_{i,j} \tag{3}$$

Similarly, the messages destined to P_i cannot be overlapped in time. Thus, these P messages cannot be completed at a time earlier than the sum of the entries in column i of **C**.

$$t_{max} \geq \sum_{j=0}^{(P-1)} \mathbf{C}_{j,i} \tag{4}$$

We can impose a similar pair of constraints on t_{max} for each of the P processors. The completion time of the schedule cannot be less than the summation of send times or receive times at any processor, whichever is larger. This is therefore a *lower bound* t_{lb} on the completion time. Thus, t_{lb} is the largest among all the row sums and column sums of the communication matrix C.

$$t_{lb} = \max_{i=0 \to (P-1)} \{\max(\sum_{j=0}^{(P-1)} \mathbf{C}_{i,j}, \sum_{j=0}^{(P-1)} \mathbf{C}_{j,i})\} \tag{5}$$

4.3 Baseline: Caterpillar Algorithm

Since the total exchange communication scheduling problem is *NP*-Complete, we have developed heuristic algorithms. As a *baseline algorithm* for performance comparison with our heuristic algorithms, we shall use the caterpillar algorithm, which is widely used in tightly coupled homogeneous systems. This generates a schedule with P steps. In step j, $(0 \leq j < P)$, each compute node $P_i(0 \leq i < P)$ sends a message to $P_{(i+j)modP}$. Such a schedule does not incur any node contention in a homogeneous system, when the message sizes and network bandwidths are uniform. This is because all the communication events have the same duration, *i.e.* all the rectangles in the timing diagram have the same height. An important disadvantage of the baseline algorithm is that it derives a fixed schedule, which is not adaptive to variations in message lengths or network performance.

We illustrate our scheduling techniques with a running example. Figure 5 shows an example communication problem, represented in the timing diagram formalism. The unscheduled communication events originating from each processor are shown in increasing order of destination processor number.

In our examples, we assume that the diagonal entries in **C** are zeroes. This is a reasonable assumption, since the time for a local memory copy operation is negligible in comparison with the time for sending messages over the heterogeneous network.

Using the baseline algorithm, the schedule shown in Figure 6 is derived. Observe that the longer communication events in the earlier steps cause the later communication steps to be delayed.

Theorem 2 (Performance Bound for the Caterpillar Algorithm) *1. The completion time t_{max} of the caterpillar schedule is always within $\frac{P}{2}$ times the lower bound t_{lb}.*

2. The above bound is tight, i.e. there exist instances where the caterpillar schedule takes $\frac{P}{2}$ times the lower bound.

Proof: We first introduce the notion of a dependence graph **DG** for a given schedule. This is a directed graph with P^2 nodes, one for each communication event. A directed edge is present from node i to node j if there exists a sequential dependency between the corresponding communication events in the schedule.

Figure 7 shows the dependence graph for the caterpillar schedule with 5 processors. Column i contains all the communication events sent from Processor P_i, in the order that they appear in the schedule. Observe that the edges in **DG** are of two kinds: (i) vertical edges between adjacent nodes in the same column, and (ii) diagonal edges between nodes in adjacent columns. A correspondence exists between the graph **DG** and the communication matrix **C**. Each node in **DG** corresponds to an entry in **C**. If a vertical edge exists between two nodes, then these correspond to entries in the same column of **C**. If a diagonal edge is present, then the nodes correspond to entries in the same row of **C**.

Each path in the **DG** for the baseline schedule contains P nodes and $P-1$ edges. The completion time is equal to the weight of the longest path in the graph. Let $t_1, t_2, \ldots, t_P$ be the nodes in the longest path. Then,

$$t_{max} = t_1 + t_2 + \ldots + t_P \tag{6}$$

We can rewrite Eq (6) as

$$t_{max} = (t_1 + t_2) + (t_3 + t_4) + \ldots + (t_{P-1} + t_P) \tag{7}$$

Since adjacent nodes in any path belong to the same row or column of **C**, and from the definition of t_{lb},

$$t_{lb} \geq (t_1 + t_2), t_{lb} \geq (t_3 + t_4), \ldots, t_{lb} \geq (t_{P-1} + t_P) \tag{8}$$

Using Eq (8) in Eq (7)

$$t_{max} \leq (t_{lb} + t_{lb} + \ldots + t_{lb})$$

$$t_{max} \leq \frac{P}{2} \times t_{lb} \tag{9}$$

To prove the tightness of the bound, consider the following communication matrix:

$$\mathbf{C} = \begin{bmatrix} \epsilon & \epsilon & 1 & 1 \\ \epsilon & 1 & 1 & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon \end{bmatrix}$$

Each dependence path consists of 4 elements. The first element is always on the diagonal. Adjacent elements in the path are either in the same row or the same column. In the former case, the i^{th} element in the path is to the immediate right of the $i - 1^{th}$ element. In the latter case, the i^{th} element is immediately above the $i - 1^{th}$ element. For this example, the critical path contains all the unit-time entries, and takes 4 units of time. The lower bound is $2 + 2\epsilon$. Here, ϵ is an arbitrarily small number.

Therefore,

$$\frac{t_{max}}{t_{lb}} = \frac{4}{2+2\epsilon} \approx 2 \tag{10}$$

□

4.4 Matching-Based Scheduling Techniques

We present two matching-based scheduling techniques for the total exchange problem. The first technique finds a series of maximum weight matchings in a bi-partite graph. We also consider the variation wherein minimum matchings are found.

Our algorithm partitions the $P \times P$ communication events into P independent steps using graph matching algorithms. For a P node system, we construct a bi-partite graph with P vertices on each side. The edge from v_i on the left side to v_j on the right side is assigned a weight equal to the time for the communication event from P_i to P_j. Thus, there are $O(P^2)$ edges in the bi-partite graph. A complete matching in such a graph consists of P edges, and corresponds to a permutation of $(P_0, \ldots, P_{P-1})$. Such a matching can therefore represent a valid communication step, without contention at any processor. Well known algorithms exist for finding a maximum weight complete matching [18]. This is identical to the linear assignment problem. Our algorithm therefore consists of the following steps:

- Find a maximum weight complete matching in the bi-partite graph.
- Delete the edges selected by the matching from the graph.
- Repeat the above two steps P times. This generates P matchings.
- In the final *compaction phase,* each of the communication events is moved back in

time to the earliest possible time when it can be executed, *i.e.*, until further movement would result in node contention at the sending or receiving side.

The complexity of finding each complete matching is $O(P^3)$. The overall complexity of the P matching steps is therefore $O(P^4)$. In the compaction phase, there are a total of $O(P^2)$ communication events. Each event must be compared with the P communication events in its previous step. Thus, the computational complexity of this phase is $O(P^3)$. The overall complexity of the scheduling algorithm is therefore $O(P^4)$.

Note that although the schedule finds the communication events step by step, the communication phase does not impose a synchronization among the processors after each step. The compaction phase ensures that a communication event will begin whenever the sending and receiving processors are both ready.

In theory, the completion time of the matching-based techniques can be $\frac{P}{2}$ times the lower bound. We can prove a result similar to part (1) of Theorem 2. In practice, the performance is significantly better, and the bound is therefore not tight. Unlike the fixed schedule derived by the baseline algorithm, our matching-based schedule is adaptive. When the lengths of the communication events change with variations in network performance, the algorithm finds a different schedule with a low completion time. Section 5 presents simulation results.

For the example of Figure 5, our adaptive maximum matching algorithm derives the schedule shown in Figure 8. The matching technique groups together communication events with similar length, thereby reducing the idle cycles. Figure 8 is an optimal schedule for this example, since there exists a processor (P_1 or P_2) which is busy during the entire schedule.

4.5 Greedy Technique

The greedy technique is an approximation to the matching technique, with a lower computational complexity. This algorithm consists of the following steps:

- Initially, the communication events originating from each processor are sorted in decreasing order of communication time. Since each processor has P destinations, this sorting phase has a complexity of $O(P^2 \log P)$.

- A series of communication steps is then composed. In each communication step, we traverse the rank ordered list of every processor, with the goal of finding a destination processor. The list is traversed from the beginning to the end, and the traversal stops when a valid destination processor is found. A valid destination is one that has not been selected by this processor in a previous step, and that is not the destination of another processor in the same step.

- If the end of the list is reached without finding a destination, the processor idles during this step, and we proceed to the next processor. Due to such incomplete steps, the total number of steps could be larger than P.

- To ensure fairness, a processor which was idle in any step will be the first to pick the destination processor in the next step. If there was no idle processor in a step, the last processor in any step will be the first in the next step.

The greedy algorithm has a computational complexity of $O(P^3)$. From the description above, it is clear that the greedy algorithm is adaptive to the lengths of the communication events.

For the previous example, the communication schedule derived by the greedy algorithm is shown in Figure 9.

4.6 Open Shop Technique

Since our communication problem has similarities to the open shop scheduling problem, we have developed a scheduling algorithm based on a heuristic derived for the open shop problem [22]. Other approximate algorithms for the open shop problem are given in [5].

Each processor is considered as two independent entities, a sender and a receiver. The following data structures are maintained by the algorithm:

- For each sender $i, 0 \leq i < P$, a set R_i of receivers is maintained. Initially, this consists of all receivers to which i must send a message. When a communication event is scheduled, the appropriate receiver is deleted from the receiver set.

- The P-element arrays *sendavail* and *recvavail* contain information about the availability of the corresponding senders and receivers. For example, the i^{th} element of *sendavail* specifies the earliest time at which sender i can participate in future send operations. All elements of both these arrays are initialized to 0.

The algorithm proceeds as follows:

- Whenever a sender i becomes available at time *sendavail[i]*, its receiver set R_i is scanned, and the earliest available receiver j is selected. The communication event from i to j is scheduled to begin at time *t=max(sendavail[i], recvavail[j]). sendavail[i]* and *recvavail[j]* are assigned the value $t + \mathbf{C}[j, i]$, since the sender i and receiver j will be busy until this time. Further, j is deleted from R_i.

- If multiple senders become available at the same time (for example, at time 0), they are processed in an arbitrary order. However, all senders that become available at time t are processed before any senders that become available at a later time. The algorithm maintains a list of senders in increasing order of their time of availability.

- Whenever a sender is finished with all its operations, it is deleted from this list. The algorithm terminates when all the senders are thus deleted.

The total number of communication events to be scheduled is $O(P^2)$. The scheduling of each event takes $O(P)$ time, since the elements of the corresponding receiver set must be scanned. The algorithm therefore runs in time $O(P^3)$.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to any of the elements in its receiver set. Idle cycles are inserted in a sender's schedule only if none of its potential receivers are available. For our running example, the schedule derived by this heuristic is shown in Figure 10.

Theorem 3 (Performance Bound for the Open Shop Heuristic) *The open shop heuristic algorithm is guaranteed to find a communication schedule whose completion time is within twice the lower bound.*

Proof: Assume, without loss of generality, that the last sender to finish all its transmissions is i. Let j be the receiver that i sends its last message to. It can be deduced that during the idle cycles in sender i's schedule, receiver j must have always been busy. If this were not the case, the algorithm would have scheduled the communication event from i to j at this time. Thus, we can conclude that the sum of the idle cycles in sender i's schedule is bounded by the total time for communication events having j as the receiver, *i.e.,* the sum of elements in row j of **C**. The completion time is the sum of the total time for send events from sender i, *i.e.,* the sum of elements in column i of **C**, and the idle cycles in sender i. Thus, the completion time is at most the sum of a row and a column in the communication matrix **C**, and is hence within twice the lower bound. □

5 Experimental Results

We have developed a software simulator that executes the scheduling algorithms discussed in Section 4, and calculates the completion time for each of them. The simulator accepts processor count and communication times as input, and generates the schedules based on these techniques. We have used this simulation tool to evaluate the baseline, maximum matching, minimum matching, greedy, and open shop scheduling techniques.

The simulator generates random performance characteristics for pairwise network performance, using information from the GUSTO directory service as a guideline. For a P processor simulation, the simulator first constructs the communication matrix **C** with P^2 elements. Element $\mathrm{C_{ij}}$ represents the network performance and message size from P_i to P_j. We construct the matrix **C** by randomly selecting elements from a set of base values. The base values are real network performance figures, obtained from the GUSTO testbed. Consider the example of a 20 processor system with 1 kB message sizes. We first evaluate the base values – the communication times to send 1 kB messages between processors in the GUSTO system. Each of the 400 elements in the desired communication matrix **C** is randomly selected from among these base values. In our experiments, we have selected message sizes of 1kB, 1MB, and a random mix of these two sizes. The base values for 1 MB messages and 1 kB messages are shown in Table 3 and Table 4, respectively. We also assume that the diagonal entries in C are zeroes. This is valid, since the time for a local memory copy operation is negligible in comparison with the time for sending messages over the heterogeneous network. The scheduling techniques are then applied to this communication matrix. Results for the different message sizes are shown in Figures 11, 12, and 13. Systems with up to 50 processors were considered. Figure 11 shows the results for messages of size 1 kB. Figure 12 shows the results for messages of size 1 MB. Figure 13 represents the situation when a mix of small and large messages are transmitted. The graphs clearly show the performance improvements that can be achieved by our communication scheduling techniques. The open shop algorithm finds schedules that are very close to the lower bound, often within 2%, and always within 10 %. The maximum and minimum matching based techniques find schedules with comparable completion times. These are within 15% of the lower bound. The schedules generated by the greedy algorithm are within 25% of the lower bound.

Figure 14 considers a scenario when some of the processors are designated as servers. The message sizes from the servers to the other (client) processors are assumed to be large. The message sizes between the servers themselves and also between the client processors are small. This is typical in multimedia applications, where images and video clips reside on servers, and are accessed by other processors. In our experiment, 20% of the processors are assumed to be servers. Data is also assumed to be partitioned over the servers, so that the load on the servers is balanced. It can be seen that the baseline algorithm performs very poorly in such scenarios. Our algorithms perform 2 to 5 times faster than the baseline in these examples. Based on our results, the open shop algorithm achieves the best performance.

6 Enhancements and Future Research

In the previous sections, we presented our approach for developing communication scheduling techniques that are adaptive to network performance variations. To the best of our knowledge, this is one of the early efforts in formalizing communication problems relevant to network-based computing. Several exciting research issues remain to be explored. In this section, we discuss some future research directions that are motivated by the need for network-aware communication scheduling.

6.1 Enhancing the Model

Our scheduling algorithms were developed using a simple yet effective communication model. In Section 3, we mentioned the assumptions made by our model, and the validity of these assumptions. Enhanced versions of the model can be formulated by relaxing some of these assumptions. For example, one restriction is that a processor can send and receive only one message at a time. This restriction can be relaxed in two ways.

When multiple messages arrive at a node, we can assume that the messages are received in an interleaved fashion. For example, the use of multithreading allows multiple simultaneous communication events in Nexus. An additional parameter α can be introduced for the overhead incurred in context switching between the multiple receiving threads. Thus, if t_1 and t_2 are the times for individually receiving two messages, the total time for receiving them simultaneously would be $(1 + \alpha)(t_1 + t_2)$.

It could also be assumed that a finite buffer space is available at nodes to receive messages. When multiple messages arrive at a node, one of the messages is received by the application, while the others are queued in the buffer. The sending nodes do not wait until the receive operation is complete, but only until the message is stored in the buffer. If the buffer is full, the sender must wait until adequate free space is created in the buffer.

6.2 Incremental Dynamic Scheduling

The communication schedules presented in Section 4 are computed at run-time based on information obtained from the directory service. In many sensor-based applications, a series of continuously arriving data sets are processed in an identical manner. In such cases, the overhead for repeatedly calculating the communication schedule at run-time can be expensive, especially when the number of processors is large. It is therefore necessary to develop scheduling techniques which have significantly lower computational costs.

An incremental approach would be one way to reduce the complexity of deriving dynamic communication schedules. Here, a communication schedule is computed once, either at compile time or during the first run-time occurrence. At each subsequent invocation, the incremental algorithm must refine this communication schedule to find a new communication schedule. The algorithm would query the directory service regarding changes in the bandwidths. In this context, the research problem is that of developing fast algorithms for refining an existing communication schedule.

6.3 Enhancing Adaptivity of the Schedules

In some scenarios, the lengths of all communication events may not be known even when the communication is started. This could happen because variations in network performance are so rapid that significant changes could occur within the duration of the communication schedule. In such cases, an initial communication schedule can be derived using estimates of the communication times. The schedule can then be modified at intermediate *checkpoints*. At these checkpoints, processors decide whether the difference between the estimated time and actual time is large enough to require rescheduling. The checkpoints could be defined in different ways: after each communication event is complete ($O(P)$ checkpoints), or after half the remaining communication events are complete ($O(\log P)$ checkpoints), and so on.

6.4 Scheduling with Critical Resources or QoS Constraints

We have discussed communication schedules where the goal is to minimize the completion time. In many scenarios, other cost measures are also important. For example, one of the processors in the heterogeneous system could be a critical resource (*e.g.,* an expensive supercomputer). The schedule should complete the communication events of this processor as early as possible, even if it delays the other processors.

Quality of Service (QoS) requirements in some applications can introduce other variations in the problem formulation. For example, data forwarding and data staging problems arise in the BADD project [7]. The QoS parameters associated with each message are deadlines and priorities. The communication schedule must ensure that data items reach their destinations by the specified real-time deadlines. When multiple communication events contend for a communication link, the scheduling algorithm must sequence them based on their respective deadlines and priorities.

7 Conclusion

In this paper, we have developed a uniform framework for communication scheduling in heterogeneous network-based systems. The framework consists of a directory service, a communication model, timing diagrams, and scheduling algorithms. We discussed our approach for the design of adaptive communication techniques, and applied it to the problem of all-to-all personalized communication. Although this problem has been thoroughly researched for homogeneous systems, we showed that well known algorithms perform poorly in the presence of network heterogeneity. We developed algorithms based on bi-partite graph matching, and a heuristic algorithms based on Open shop scheduling. We showed that our algorithms performed significantly better than a well known homogeneous communication scheduling algorithm. Our algorithms are adaptive and execute at run-time, based on network performance information obtained from the directory service. To the best of our knowledge, this is one of the early efforts in formalizing communication problems in a distributed heterogeneous computing environment. Our paper also discusses several new research problems related to communication scheduling in network-based systems, that arise due to the unique features of such environments. These include communication scheduling with QoS constraints and techniques to reduce the complexity of the scheduling algorithm.

Acknowledgments

We thank Roy Jonker of MagicLogic Optimization, Inc. for his public domain program to solve the Linear Assignment Problem. This routine was used to find the matchings in our algorithms. We also thank Craig Lee and Paul Stelling of The Aerospace Corporation and the members of the MSHN project for helpful technical suggestions. We are also thankful to the reviewers for their suggestions.

References

[1] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *IEEE Trans. Parallel and Distributed Systems*, 6(2):154–164, February 1995.

[2] P. B. Bhat. *Communication Scheduling Techniques for Distributed Heterogeneous Systems.* PhD thesis, University of Southern California, 1999.

[3] Prashanth B. Bhat, Viktor K. Prasanna, and C.S. Raghavendra. Block-cyclic redistribution over heterogeneous networks. In *Proc. ISCA Intl. Conf. Parallel and Distributed Computing Systems*, pages 242–249, Sep. 1998.

[4] Prashanth B. Bhat, C.S. Raghavendra, and Viktor K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *Proc. IEEE Intl. Conf. Distributed Computing Systems*, June 1999.

[5] H. Brasel, T. Tautenhahn, and F. Werner. Constructive heuristic algorithms for the open shop problem. *Computing*, 51:95–110, 1993.

[6] P. Brucker. *Scheduling Algorithms.* Springer, 1995.

[7] DARPA ISO Web Page for the BADD Program. *https:// www.iso.darpa.mil/ WD@27000.cgi?get+iso:: Office+Information_System +WDI_i_home_frames.*

[8] Tony DeWitt, Thomas Gross, Bruce Lowekamp, Nancy Miller, Peter Steenkiste, Jaspal Subhlok, and Dean Sutherland. ReMoS: A resource monitoring system for network-aware applications. Technical Report CMU-CS-97-194, School of Computer Science, Carnegie Mellon University, Dec 1997.

[9] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewskiand, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Intnl. Symp. on High Performance Distributed Computing*, pages 365–375, 1997.

[10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[11] Globus Web Page. *http://www.globus.org.*

[12] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.

[13] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Wide-area computing: Resource sharing on a large scale. *Computer*, 32(5):29–37, 1999.

[14] D. Hensgen et al. An overview of MSHN: The Management System for Heterogeneous Networks. In *Proc. Heterogeneous Computing Workshop*, pages 184–198, Apr. 1999.

[15] JunSeong Kim and David J. Lilja. Exploiting multiple heterogeneous networks to reduce communication costs in parallel programs. In *Proc. Heterogeneous Computing Workshop*, pages 83–95, April 1997.

[16] JunSeong Kim and David J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intl. Symp. High Performance Distributed Computing*, 1997.

[17] H. Korab and M. D. Brown, *Eds. Virtual Environments and Distributed Computing at SC '95: GII Testbed and HPC Challenge Applications on the I-WAY.* ACM/IEEE Supercomputing '95, 1995.

[18] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[19] Legion Web Page. *http://legion.virginia.edu.*

[20] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient algorithms for block-cyclic redistribution of arrays. *Algorithmica*, to appear.

[21] MSHN Web Page. *http://www.mshn.org.*

[22] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, June 1994.

[23] L. Smarr and C. E. Catlett. Metacomputing. *Commns. of the ACM*, 35(6):45–52, June 1992.

[24] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, March 1998.

[25] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, I. Ra, D. Kim, Y. Kim, X. Bing, and B. Ye. The software architecture of a virtual distributed computing environment. In *Proc. Sixth IEEE Intl. Symp. on High Performance Distributed Computing*, 1997.

[26] C.-L. Wang, P. B. Bhat, and V. K. Prasanna. High-performance computing for vision. *Proceedings of the IEEE*, 84(7):931–946, July 1996.

Author Biographies

Prashanth B. Bhat is a Ph.D. candidate in Computer Engineering at the University of Southern California, Los Angeles. He received his B.Tech. degree in Computer Engineering from the Karnataka Regional Engineering College, India, in 1992. He received his M.E. degree in Computer Science and Engineering from the Indian Institute of Science, Bangalore, in 1994. During the summer of 1998, he was a research intern at Hewlett-Packard laboratories, Palo Alto. His research interests include scheduling techniques for parallel and distributed systems, High Performance Computing and parallel computer architecture.

Viktor K. Prasanna (V.K. Prasanna Kumar) received his B.S. in Electronics Engineering from the Bangalore University and his M.S. from the School of Automation, Indian Institute of Science. He obtained his Ph.D. in Computer Science from Pennsylvania State University in 1983. Currently, he is a Professor in the Department of Electrical Engineering - Systems, University of Southern California, Los Angeles and serves as the Director of the Computer Engineering Division. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision.

Dr. Prasanna has published extensively and consulted for industries in the above areas. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He was the general chair of the IEEE International Parallel Processing Symposium, 1998 and has been the key person in developing the meeting into a premier international meeting in parallel computing. He also serves on the editorial boards of the Journal of Parallel and Distributed Computing and

IEEE Transactions on Computers. He is the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a Fellow of the IEEE.

C. S. Raghavendra is with The Aerospace Corporation in El Segundo, California. From September 1982 to December 1991 he was on the faculty of Electrical Engineering-Systems Department at the University of Southern California, Los Angeles. From January 1992 to August 1997 he was on the faculty of the School of Electrical Engineering and Computer Science at the Washington State University in Pullman as the Boeing Centennial Chair Professor of Computer Engineering. He received the B.E and M.E degrees in Electronics and Communication from the Indian Institute of Science, Bangalore, in 1976 and 1978, respectively. He received the Ph.D degree in Computer Science from the University of California at Los Angeles in 1982. His research interests are wireless networks, parallel processing, fault tolerant computing, and distributed systems. Dr. Raghavendra is a recipient of the Presidential Young Investigator Award for 1985 and became a Fellow of the IEEE in 1997.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		34.5	89.5	12	42
ANL	34.5		20	26.5	4.5
IND	89.5	20		42.5	21.5
USC-ISI	12	26.5	42.5		29.5
NCSA	42	4.5	21.5	29.5	

Table 1: Latency (ms) between 5 GUSTO sites.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		512	246	2044	391
ANL	512		491	693	2402
IND	246	491		311	448
USC-ISI	2044	693	311		4976
NCSA	391	2402	448	4976	

Table 2: Bandwidth (kbits/s) between 5 GUSTO sites.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		15660	32610	3926	20502
ANL	15660		16313	11570	3335
IND	32610	16313		25766	17879
USC-ISI	3926	11570	25766		1637
NCSA	20502	3335	17879	1637	

Table 3: Communication times(ms) for 1 Mbyte messages between 5 GUSTO sites.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		50	122	16	62
ANL	50		36	38	8
IND	122	36		68	39
USC-ISI	16	38	68		31
NCSA	62	8	39	31	

Table 4: Communication times(ms) for 1 kbyte messages between 5 GUSTO sites.

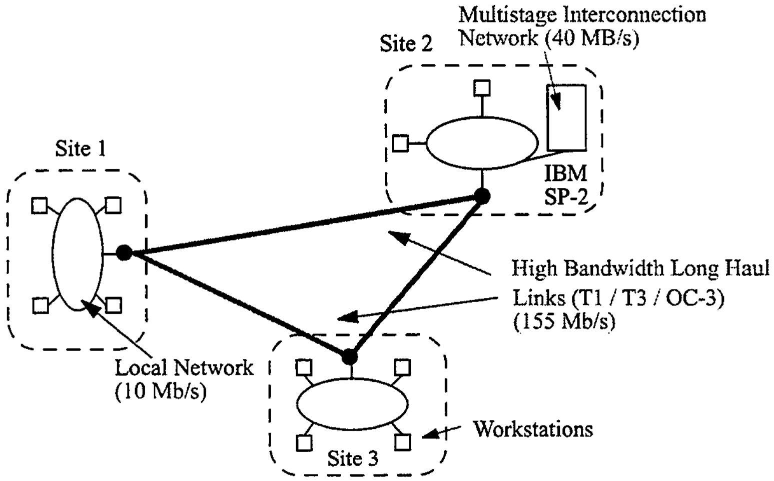

Multistage Interconnection Network (40 MB/s)
Site 2
Site 1
IBM SP-2
High Bandwidth Long Haul Links (T1 / T3 / OC-3) (155 Mb/s)
Local Network (10 Mb/s)
Workstations
Site 3

Figure 1: A typical metacomputing system.

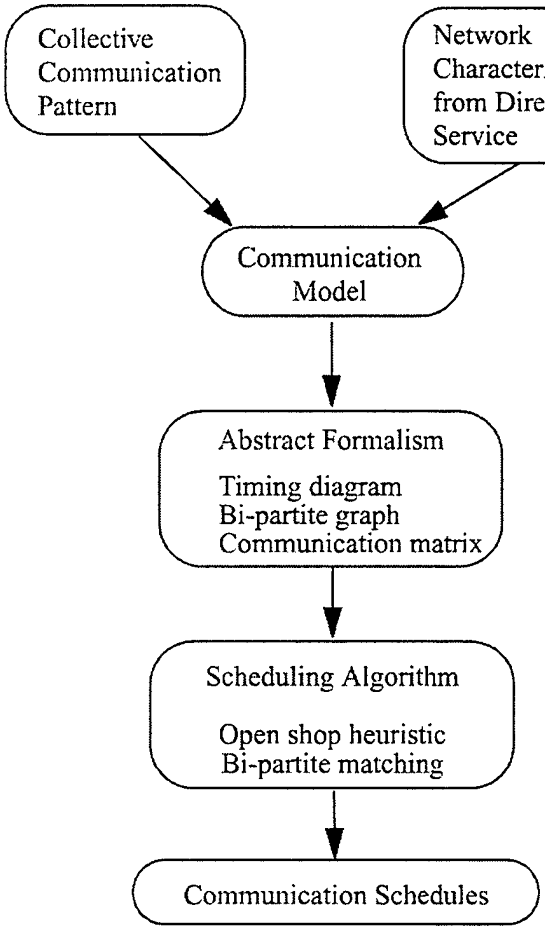


Figure 2: Our communication scheduling approach.

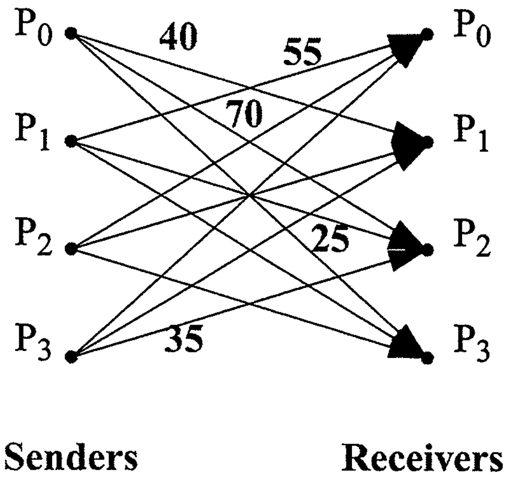


Figure 3: An example of the bi-partite graph formalism.

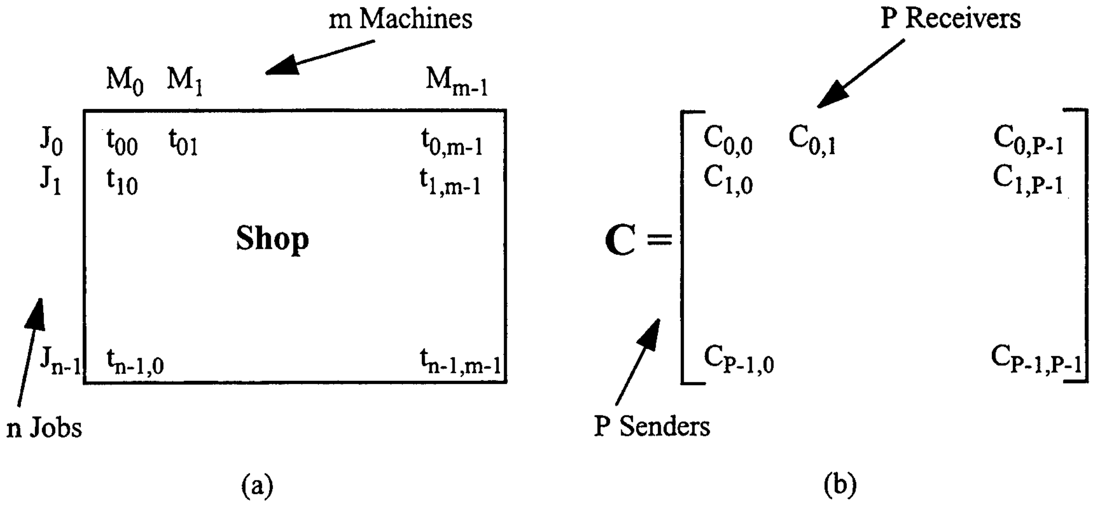


Figure 4: Equivalance of the open shop and *TOT_EXCH* problems.

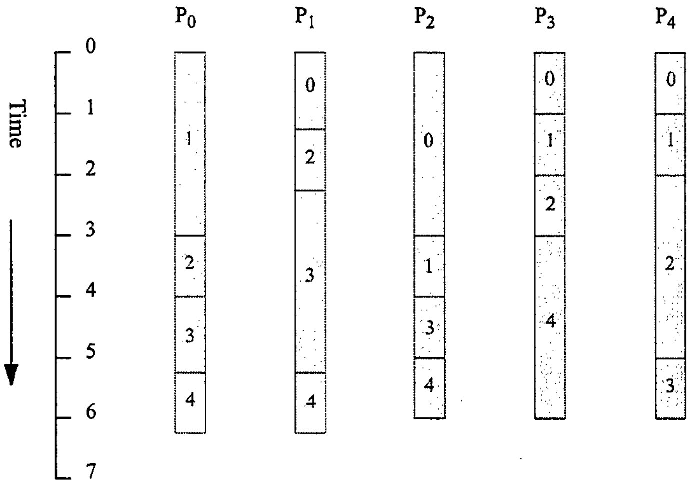


Figure 5: Example problem.

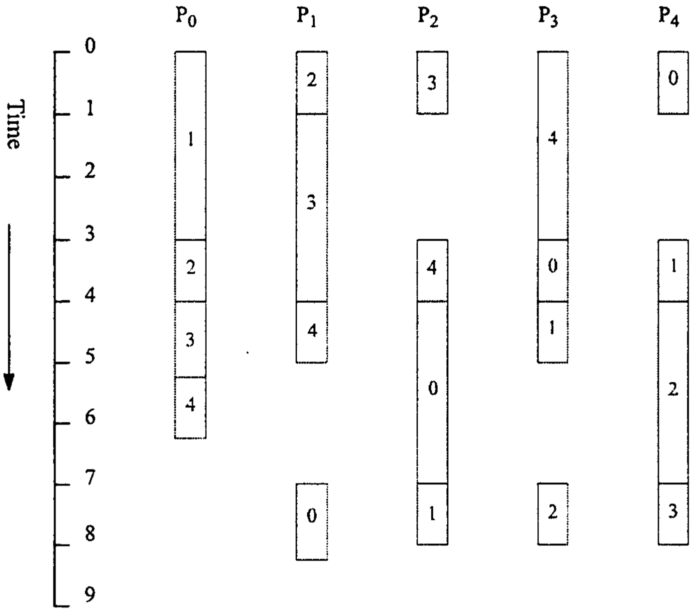


Figure 6: Schedule generated by baseline algorithm.

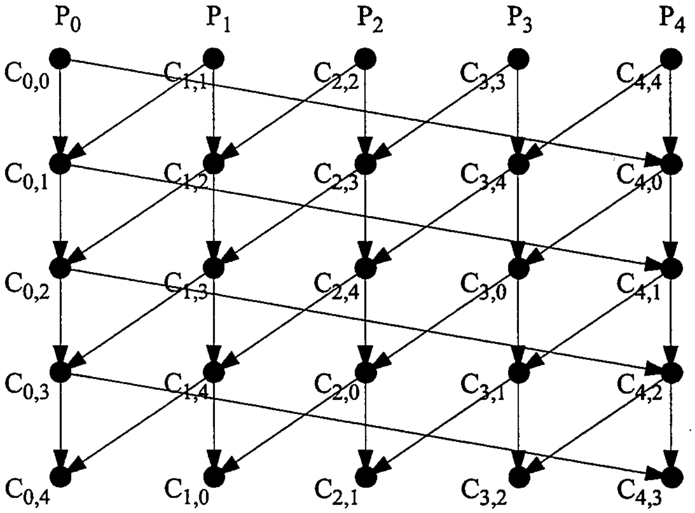


Figure 7: Dependence graph for the baseline schedule.

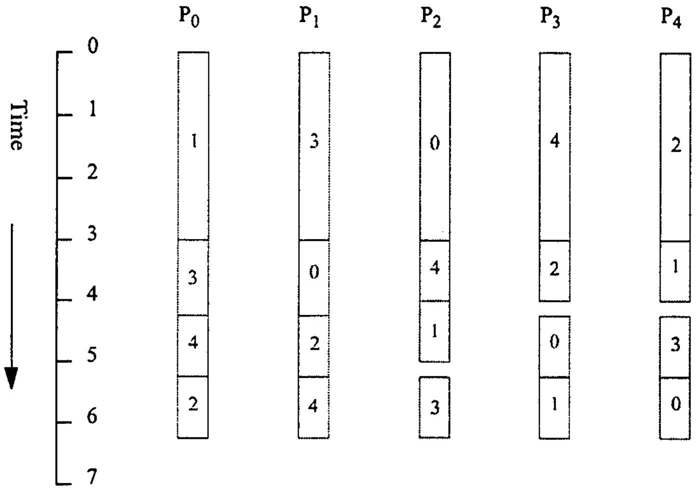


Figure 8: Schedule generated by a series of maximum matchings.

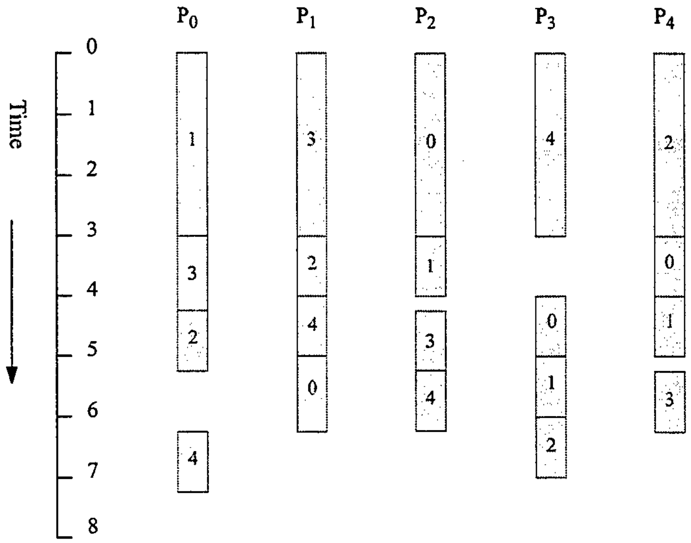


Figure 9: Schedule generated by the greedy algorithm.

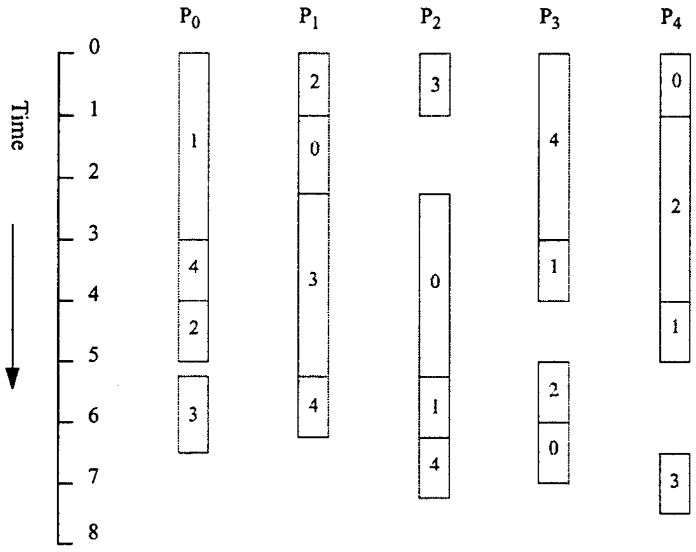


Figure 10: Schedule generated by the open shop algorithm.

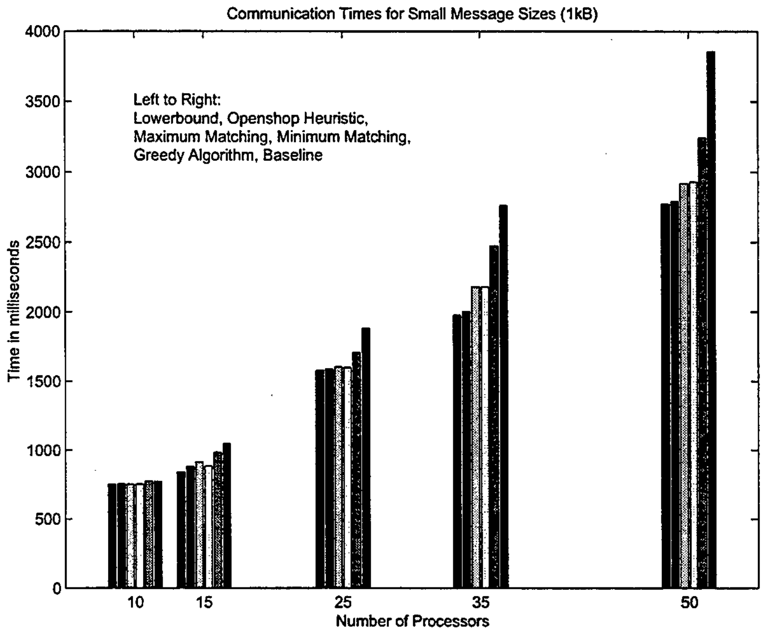


Figure 11: Simulator results for all-to-all personalized communication with small message sizes.

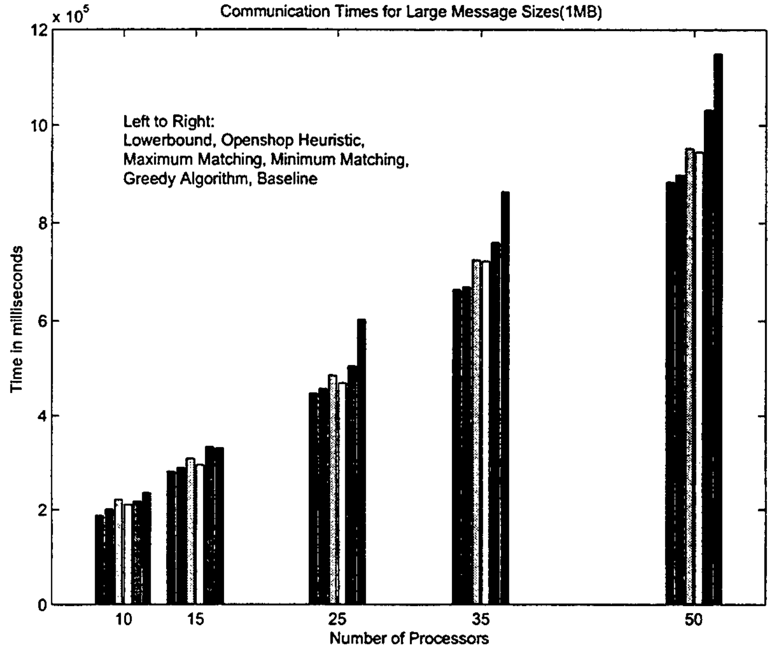


Figure 12: Simulator results for all-to-all personalized communication with large message sizes.

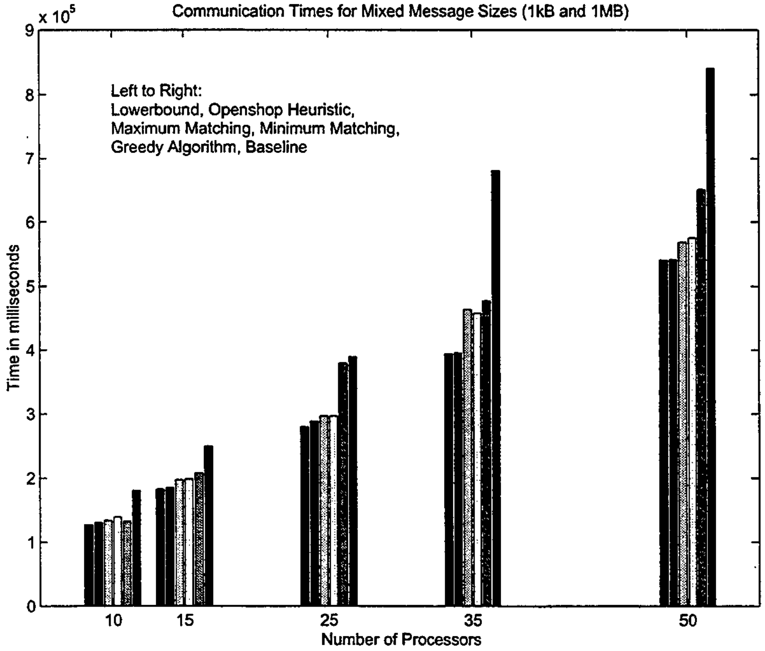


Figure 13: Simulator results for all-to-all personalized communication with mixed message sizes.

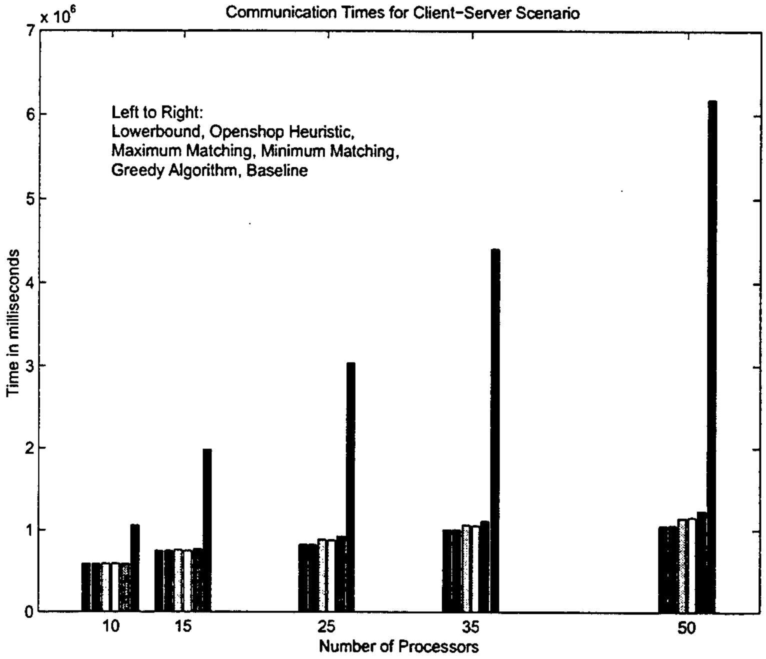


Figure 14: Simulator results for all-to-all personalized communication when 20% of the processors are servers. Servers send large messages to their clients.

Figure 1: A typical metacomputing system.

Figure 2: Our communication scheduling approach.

Figure 3: An example of the bi-partite graph formalism.

Figure 4: Equivalance of the open shop and *TOT_EXCH* problems.

Figure 5: Example problem.

Figure 6: Schedule generated by baseline algorithm.

Figure 7: Dependence graph for the baseline schedule.

Figure 8: Schedule generated by a series of maximum matchings.

Figure 9: Schedule generated by the greedy algorithm.

Figure 10: Schedule generated by the open shop algorithm.

Figure 11: Simulator results for all-to-all personalized communication with small message sizes.

Figure 12: Simulator results for all-to-all personalized communication with large message sizes.

Figure 13: Simulator results for all-to-all personalized communication with mixed message sizes.

Figure 14: Simulator results for all-to-all personalized communication when 20% of the processors are servers. Servers send large messages to their clients.

Block-Cyclic Redistribution over Heterogeneous Networks *

Prashanth B. Bhat[a] Viktor K. Prasanna[a] C. S. Raghavendra[b]

[a] *Department of EE-Systems, University of Southern California, Los Angeles, CA 90089-2562.*
Email: {prabhat,prasanna}@halcyon.usc.edu
[b] *The Aerospace Corporation, Los Angeles, CA 90009. Email: raghu@aero.org*

Abstract

Clusters of workstations and networked parallel computing systems are emerging as promising computational platforms for HPC applications. The processors in such systems are typically interconnected by a collection of heterogeneous networks such as Ethernet, ATM, and FDDI, among others. In this paper, we develop techniques to perform block-cyclic redistribution over P processors interconnected by such a collection of heterogeneous networks.

We represent the communication scheduling problem using a timing diagram formalism. Here, each interprocessor communication event is represented by a rectangle whose height denotes the time to perform this event over the heterogeneous network. The communication scheduling problem is then one of appropriately positioning the rectangles so as to minimize the completion time of all the communication events. For the important case where the block size changes by a factor of K, we develop a heuristic algorithm whose completion time is atmost twice the optimal. The running time of the heuristic is $O(PK^2)$.

Our heuristic algorithm is adaptive to variations in network performance, and derives schedules at run-time, based on current information about available network bandwidth. Our experimental results show that our schedules always have communication times that are very close to optimal.

Keywords: Workstation clusters, heterogeneous networks, communication scheduling, block-cyclic redistribution.

* A preliminary version of this paper appears in the Proceedings of the 11th ISCA International Conference on Parallel and Distributed Computing (PDCS 1998).

1. Introduction

Due to advances in high-speed networks, workstation clusters and loosely connected distributed systems are being used as platforms for High Performance Computing. Wide area networking technology has also enabled the development of *metacomputers* [13], wherein grand challenge applications are parallelized across geographically distributed supercomputers and visualization devices. Such distributed systems are typically interconnected with a collection of many different kinds of communication networks, such as ATM, HiPPI, and Ethernet.

Prototype systems with such heterogeneous networks have been built. For example, [6,5] evaluated the performance of HPC applications on a cluster of workstations interconnected with ATM, Ethernet, and FDDI networks. The performance characteristics of each of the networks were first evaluated by sending messages of various sizes over the particular network. These characteristics were then used to choose a suitable technique for data communication. The *Performance Based Path Selection (PBPS)* technique selects one of the networks for a given communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks.

The I-WAY (Information Wide Area Year) metacomputer at SC '95 [7] consisted of over 10 networks of varying bandwidths, protocols, and routing technology. The HiPer-D project investigates the use of networked distributed computing capabilities in battle management systems on U.S. Navy cruisers. The Battlefield Awareness and Data Dissemination (BADD) program develops techniques for delivering multimedia data to mobile troops over a combination of wired and wireless networks [14].

From the above examples, it is clear that heterogeneity is a salient characteristic of the interconnection network in local area clusters and distributed computational environments. Further, the network is shared among multiple applications. The performance therefore depends upon the current traffic conditions, and typically varies over time.

For scalable performance on such a platform, support for fast application-level communication is necessary. Efficient implementations of important collective communication kernels must be incorporated into communication libraries. In this paper, we develop communication techniques for *block-cyclic redistribution* over such heterogeneous networks. We consider the important case where the block size changes by a factor of K. Our techniques can also be extended to other redistribution problems.

The block-cyclic distribution is widely used in many HPC applications to partition an array over multiple processors. For example, in signal processing applications, the block-cyclic distribution is the natural choice for radar and sonar data cubes. Many of the frequently occurring communication patterns, such as the corner turn operation, can be then viewed as block-cyclic redistribution operations [9]. ScaLAPACK, a widely used mathematical software for dense linear algebra computations, also uses a block-cyclic distribution for good load balance and computational efficiency. Matrix transpose operations, which often occur in linear algebra computations, are a special case of the block-cyclic redistribution. HPF provides directives for specifying block-cyclic distribution and redistribution of arrays.

The problem of block-cyclic redistribution in a tightly-coupled homogeneous parallel system has been well researched. However, the heterogeneity and sharing of the network make it necessary to develop new communication scheduling techniques. In Section 4, we present a communication scheduling algorithm that is well suited for heterogeneous networks. The algorithm is adaptive to variations in network performance. The schedule is derived at run-time, based on current information about network load.

Our scheduling approach is based on a communication model that represents the communication performance between every processor pair using two parameters: a start-up time and a data transmission rate. We formalize the communication scheduling problem using a timing diagram representation. Each interprocessor communication event is represented as a rectangle whose height equals the time to perform the communication over the heterogeneous network. The height is calculated using our communication model. The communication scheduling problem is then one of appropriately positioning the rectangles in the timing diagram so as to minimize the completion time of all the communication events. Our heuristic algorithm derives a communication schedule whose completion time is always within twice the optimal. The running time of the heuristic is $O(PK^2)$, where P is the number of processors, and K is the factor by which the block size changes.

The rest of the paper is organized as follows. Section 2 discusses the characteristics of the block-cyclic redistribution communication pattern. Section 3 discusses some previous research efforts on block cyclic redistribution. Section 4 introduces our communication model for the heterogeneous network and presents our heuristic algorithm for block-cyclic redistribution. Section 5 presents performance results from the experimental implementation of our algorithm. Section 6 concludes the paper and discusses future research directions.

2. The Block-Cyclic Redistribution Problem

The block-cyclic distribution of an array can be defined as follows [16]: given P processors, an array with N elements, and a block size x, the distribution first partitions the array elements into contiguous blocks of x items each. b_i is the i^{th} block, $0 \leq i < \frac{N}{x}$ [1]. The blocks are then assigned to processors in a round robin fashion so that b_i is assigned to processor $(i \bmod P)$. We denote a block-cyclic distribution of block size x as *cyclic*(x).

The block-cyclic data redistribution problem consists of reorganizing an array from one block-cyclic distribution to another. The most frequently encountered version of this redistribution problem is the *cyclic*(x) to *cyclic*(Kx) redistribution, which is the problem we consider in this paper. We denote the *cyclic*(x) to *cyclic*(Kx) redistribution among P processors as $\Re_x(K, P)$.

Figure 1 shows the example of $\Re_2(3, 4)$. The array **A** which has $N = 48$ elements is shown in Figure 1(a). Figure 1(b) shows the initial distribution, *cyclic*(2). Here, the blocks are of size 2 elements, and are labeled as $b_0, b_1, \ldots, b_{23}$. The blocks are assigned to $P(= 4)$ processors in a round robin fashion. If the block size is increased by a factor of $K(= 3)$, *i.e.*, the new block size becomes 6, each set of three consecutive blocks becomes a new block, as shown in Figure 1(c) and (d).

Block-cyclic redistribution consists of three main phases:

1. **Index set computation and message generation:** Each processor computes indices of array elements that are to be communicated with the other processors, as well as the destination processors of such array elements. The elements are then packed into message buffers, one for each destination processor.

2. **Communication scheduling:** A given processor contains messages for a total of K processors, and will also receive messages from K processors. The aim of communication scheduling is to reduce the overall communication

[1] For simplicity, we assume that x divides N.

time. During this phase, each processor determines an ordering among its send and receive events, so as to reduce contention. Recall that node contention occurs when multiple processors simultaneously send messages to a receiver.

3. **Interprocessor communication:** The processors send and receive messages in the order specified by the communication schedule. This phase incurs software start-up overheads for invocation of the send and receive system calls, and transmission costs for sending data over the interconnection network. In the absence of communication scheduling, this phase can become very inefficient due to node contention.

The interprocessor communication pattern of $\Re_x(K, P)$ can be represented by a communication graph, shown in Figure 2(a). Each edge represents a message that is to be sent between the corresponding processors. Note that each processor $P_i, 0 \leq i < P$ must send messages to K destinations, and receive messages from K sources. For given values of x, K, and P, the position of edges in this graph and the array indices corresponding to each edge are computed during the index computation phase. In [8], we have developed efficient techniques for index computation, for systems with homogeneous networks.

Figure 2(b) shows an example of a K-step communication schedule for $\Re_x(3, 4)$. Here, the communication pattern of Figure 2(a) is broken down into a series of contention-free communication steps. When the network is homogeneous, all the communication events within any step of Figure 2(b) would take the same amount of time. In [8], we have developed communication scheduling techniques for such a homogeneous scenario. However, when the network links are heterogeneous, the time taken for each message varies with the available network bandwidth between the corresponding processors. Due to this non-uniformity in message communication times, node contention and idle cycles would be introduced in the schedule of Figure 2(b) if it was used without modification. It is therefore necessary to develop new communication scheduling techniques for cyclic redistribution over heterogeneous networks. Section 4 presents our new algorithms for this problem.

3. Related Work

The block-cyclic redistribution problem has been the focus of several research efforts. Techniques have been developed for both the index computation phase and the communication scheduling phase over a homogeneous network. In [15], Choudhary *et. al.* present efficient index computation algorithms for $\Re_x(K,P)$, when $P \bmod K = 0$. They also consider the redistribution from *cyclic*(x) to *cyclic*(y), for general x and y, using *gcd* and *lcm* methods.

In [11], Banerjee *et. al.* represent a *cyclic*(x) distribution as a set of strided line segments. Using this formalism, the array elements to be exchanged between a pair of processors is computed by the intersection of the respective line segments.

Sadayappan *et. al.* [4] and Walker *et. al.* [16] have developed algorithms for the communication scheduling phase. Here, a K step schedule is given for $\Re_x(K,P)$. At each step, processors exchange data in a contention-free manner: each processor sends data to exactly one processor and receives data from exactly one processor.

In [8], we introduced a uniform framework to develop redistribution algorithms for $\Re_x(K,P)$. Based on this framework, efficient algorithms were developed for reducing both the index computation and communication overheads. Three classes of techniques were presented for $\Re_x(K,P)$: direct, indirect, and hybrid. In the direct approach, a block is sent directly from a source processor to its destination without being sent to intermediate processors. The direct approach performs the $\Re_x(K,P)$ communication in K communication steps. The indirect approach performs $\Re_x(K,P)$ in atmost $\lceil \log_2 K \rceil + 2$ steps. Here, the array elements are communicated in a "combine and forward" manner. The hybrid approach is a combination of the direct and indirect approaches.

In [1], communication schedules are developed for the general redistribution problem of *cyclic*(r) over a set of P processors, to *cyclic*(s), over a different set of Q processors. Graph matching algorithms are used to develop communication schedules in this work. These techniques have two important drawbacks: (i) The communication scheduling phase is expensive, due to the use of graph matching algorithms, and (ii) All processors are synchronized after each step in the interprocessor communication phase. This increases the interprocessor communication time.

[10] considers the problem of run-time redistribution from *cyclic*(x) on P processors to *cyclic*(Kx) on Q processors, over a homogeneous network. The

algorithm is based on a generalized circulant matrix formalism. The generated schedule minimizes the number of communication steps and eliminates node contention in each communication step. The network bandwidth is fully utilized by ensuring that equal-sized messages are transferred in each communication step.

Performance studies of heterogeneous networks were reported in [6]. Experiments were performed on a local cluster of workstations, interconnected with ATM, Ethernet, and Fibre-Channel networks. The performance characteristics of each of the networks were first evaluated by sending messages of various sizes over the particular network. These characteristics were used to choose a suitable technique for communication over the heterogeneous network. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks. However, this research only considered point-to-point communication between a pair of nodes in the system. In comparison, our paper investigates the collective communication pattern of block-cyclic redistribution.

The Management System for Heterogeneous Networks (MSHN) project at Naval Postgraduate School, USC, and Purdue University is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. The goal is to schedule processor and network resources among individual applications so that their QoS requirements are satisfied. In this context, data staging techniques for distributed systems with heterogeneous networks have been considered [14].

4. Our Communication Scheduling Approach

As discussed in Section 2, block-cylic redistribution consists of index computation, communication scheduling, and interprocessor communication. Since the index computation phase is independent of the network characteristics, techniques developed for homogeneous networks [8] can be used. In this section, we consider the communication scheduling phase. We first discuss the assumptions and communication model that we shall use to analyze our communication schedule. Section 4.3 presents our communication scheduling algorithm.

4.1. Communication Model

The overall network in the P processor system consists of several heterogeneous network components. Each component interconnects a subset of the processors. We can model such a network as a completely connected virtual network with heterogeneous performance between each pair of processors. Thus, the path between any pair of processors can be modeled as a single link, with the effective performance of the heterogeneous path. Techniques to aggregate the performance of different networks into a single virtual network have been considered, and are an active area of research [6]. We model the communication performance of the path between a pair of processors P_i and P_j by a start-up cost $T_{i,j}$ and a data transmission rate $B_{i,j}$. Thus, to send a m byte message between P_i and P_j, the time taken is $T_{i,j} + \frac{m}{B_{i,j}}$. When the message sizes are large, the data transmission cost is the dominating component, and the start-up cost can be ignored [2]. Typical values for the start-up cost could be in the range of 10 to 50 μ s, while typical values for the bandwidth could be in the range of a few Mb/s to hundreds of Mb/s.

We assume that the effective network performance between any pair of processors will not change during the communication phase. This can be ensured if the application reserves network bandwidth for the duration of the communication. [2] and [17] discuss issues relating to reserving network resources.

We assume that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple distinct messages to be transmitted simultaneously. If multiple nodes simultaneously send to any node P_j, these messages are received one after the other at P_j. We say that node contention occurs at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation. We do not consider the use of wild-card non-blocking receives. Although this can allow a processor to simultaneously wait for many receives, large buffer space overheads are incurred.

[2] We make this approximation in our experiments.

4.2. Timing Diagrams

Communication schedules can be conveniently represented by timing diagrams. An example of a timing diagram for $\Re_x(3,4)$ is shown in Figure 3. A timing diagram consists of P columns, one per processor. The vertical axis represents time. The communication events in column i represent the messages sent from processor P_i. The rectangle labeled j in column i represents the message sent from P_i to P_j [3]. The height of the rectangle denotes the time for the communication event. The width of the rectangle does not have any significance. Once the message sizes and T_i and B_i parameters for all processors are known, the heights of all the rectangles can be determined. Thus, the timing diagram inherently absorbs the heterogeneity in network parameters and message lengths.

Any communication scheduling algorithm must determine the positions of the communication events so that an efficient schedule is found. The goal is to find the schedule that has the minimum *completion time, i.e.,* the time at which the last communication event is completed. Since a node cannot send multiple messages simultaneously, no overlap is allowed among any of the rectangles in a column. Similarly, since multiple simultaneous receive events are not permitted at a processor, all the rectangles with the same label j must have mutually disjoint time intervals. Thus, the completion time of the schedule cannot be less than the summation of send times or receive times at any processor, whichever is larger. This quantity is therefore a *lower bound* on the completion time of any schedule.

4.3. Our Scheduling Algorithm

During the index computation and communication scheduling phases, each processor independently computes the entire schedule. The communication matrix **C**, which represents the time for each point-to-point communication event, is computed based on the values of P, K, N, and the network performance parameters. $\mathbf{C}_{i,j}$ is the height of the rectangle labeled i in column j of the timing diagram.

It can be shown that the problem of finding the optimal communication schedule is NP-complete. We have therefore developed a heuristic algorithm for this problem. Each processor is considered as two independent entities, a sender and a receiver. The following data structures are maintained by the algorithm:

[3] A receive schedule can be similarly constructed, where the communication events in column i represent messages received by processor P_i.

- For each sender $i, 0 \leq i < P$, a set R_i of receivers is maintained. These are the receivers to which i must send a message sometime during the schedule. For the $\Re_x(K, P)$ communication pattern, each set R_i will initially consist of K elements. The R_i's are obtained from the communication matrix in a straightforward way. These are the row indices of the non-zero elements in column i of $\mathbf{C}$.
- The P-element arrays *sendavail* and *recvavail* contain information about the availability of the corresponding senders and receivers. For example, the i^{th} element of *sendavail* specifies the earliest time at which sender i can participate in future send operations. All elements of both these arrays are initialized to 0.

The algorithm proceeds as follows:

- Whenever a sender i becomes available at time *sendavail[i]*, its receiver set R_i is scanned, and the earliest available receiver j is selected. The communication event from i to j is scheduled to begin at time *t=max(sendavail[i], recvavail[j])*. *sendavail[i]* and *recvavail[j]* are assigned the value $t + \mathbf{C}[j, i]$, since the sender i and receiver j will be busy until this time. Further, j is deleted from R_i.
- If multiple senders become available at the same time (for example, at time 0), they are processed in an arbitrary order. However, all senders that become available at time t are processed before any senders that become available at a later time. The algorithm maintains a list of senders in increasing order of their time of availability.
- Whenever a sender is finished with all its operations, it is deleted from this list. The algorithm terminates when all the senders are thus deleted.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to one of the elements in its receiver set. Idle cycles are inserted in a sender's schedule only if none of its potential receivers are available. Our algorithm is also *adaptive* to changes in network performance. The derived communication schedule depends on the entries of $\mathbf{C}$, which are in turn dependent on network load conditions. The schedule can be derived at runtime, using current values of network performance parameters. Previous redistribution algorithms for homogeneous networks do not provide such adaptivity. A "fixed" communication schedule is used irrespective of network load and bandwidth.

The total number of communication events to be scheduled is PK. The scheduling of each event takes $O(K)$ time, since the elements of the corresponding receiver set must be scanned. Our scheduling algorithm therefore runs in $O(PK^2)$ time. On a single node of the Cray T3E, our algorithm executed in about 10 ms for $P = 64$ and $K = 63$. This is the cost of the communication scheduling phase.

Based on the schedule, the processors perform send and receive operations during the interprocessor communication phase. Initially, a single non-blocking receive and a non-blocking send is posted by each processor. If the receive finishes first, the next receive operation is immediately posted. If the send finishes first, the next send operation is initiated. This continues until all the communication events have been completed. The cost of the interprocessor communication phase depends upon the completion time of the schedule.

Lemma 1. The heuristic algorithm is guaranteed to find a communication schedule whose completion time is within twice the optimal.

Proof. Assume, without loss of generality, that the last sender to finish all its transmissions is i. Let j be its last receiver in the schedule, *i.e.,* j is the receiver that i sends its last message to. It can be deduced that during the idle cycles in sender i's schedule, receiver j must have been always busy. If this were not the case, the greedy algorithm would have scheduled the communication event from i to j at this time. Thus, we can conclude that the sum of the idle cycles in sender i's schedule is bounded by the total communication time for events incident at receiver j, *i.e.,* the sum of elements in row j of $\mathbf{C}$. The completion time is the sum of idle cycles in sender i and the total time for send events from sender i, *i.e.,* the sum of elements in column i of $\mathbf{C}$. Thus, the completion time is at most the sum of a row and a column in the communication matrix $\mathbf{C}$, and is hence within twice the lower bound. □

Our communication problem has some similarities to the open shop scheduling problem [3]. Here, a set of M machines execute a set of N jobs. Each job J_i has a task to be executed on every machine M_j, the execution time for which is given by $t_{i,j}$. The tasks may be executed in any order. However, atmost one machine may process a given job at any time. Also, atmost one job may be processed by a given machine at any time. The goal is to schedule the tasks on the machines to minimize the completion time. The problem is known to be

NP-complete, and an algorithm similar to the above greedy heuristic has been used [12].

5. Experimental Results

In this section, we evaluate the performance of the open shop heuristic scheduling technique by measuring the time for the interprocessor communication phase. We compare this with the time for the interprocessor communication phase of the direct communication schedule, which is a fixed communication schedule. For $\Re_x(K, P)$, the direct schedule breaks down the communication pattern into K contention-free steps. It has been shown to be effective in systems with homogeneous networks, when each communication event takes the same amount of time. It can be derived from the formalisms of [4], [8], and [16].

We have developed a simulator that accepts as input the network performance parameters, and the parameters of the cyclic redistribution problem. The simulator then derives the communication schedule, and calculates the expected completion time. We also implement the communication schedules on a Cray T3E, and measure the time taken to perform the redistribution. Our experimental methodology is summarized in the following key steps:

1. For an input value of P, our simulator first generates performance parameters for the heterogeneous network. The simulator accepts as input the minimum and maximum bandwidth values, and randomly generates bandwidth values in this range for every processor pair.

2. For given input values of K and N, the communication matrix $\mathbf{C}$ is then generated. The lower bound on any communication schedule is calculated as the maximum sum over all the rows and columns of $\mathbf{C}$.

3. Our heuristic open shop scheduling algorithm is then executed, and the sender and receiver schedules at each processor are computed. The simulator also computes the estimated completion time of the schedule.

4. We implement the schedule generated in Step 3, on a Cray T3E. Although the network of the Cray T3E is homogeneous, we can simulate the heterogeneous network of Step 1 by scaling the sizes of the messages appropriately. Thus, if the heterogeneous network has a bandwidth of B_1 between nodes i and j, and if B_2 is the node-to-node bandwidth of the Cray T3E, then we scale

the message size between node i and j by a factor $\frac{B_2}{B_1}$. Each point-to-point event in the communication schedule is implemented by using MPI send and receive calls.

5. We also implement the direct schedule on the Cray T3E. The communication performance of our heuristic schedule is compared with that of the direct schedule, for various values of P and K. The communication times are measured and tabulated.

Figures 4 – 11 compare the interprocessor communication time of the open shop heuristic schedule and the direct schedule. These times were measured by our experimental implementations of both schedules on the Cray T3E. The simulated heterogeneous network had node-to-node bandwidths in the range of 10 MB/s to 200 MB/s. The Cray T3E has a bandwidth of 150 MB/s, which is the value we use for B_2 in Step 4 above. In Figure 4, results for 16 processors are shown. K is varied from a small number to $P - 1$. Experimental results for other values of P are shown in Figure 5 – 11. From these experimental results, we can conclude that our heuristic schedule achieves significant and consistent performance improvements over the direct schedule. The communication time of the open shop heuristic schedule is lower than that of the direct schedule by 10% to 60%.

We observe that, for a fixed value of P, the communication time for both the schedules varies considerably with the value of K. This phenomenon is not observed in [8], where the direct schedule is implemented on a homogeneous network. The variation occurs due to the heterogeneity in the network. The communication matrix for $\Re_x(K, P)$ has PK non-zero entries, while other entries are zeroes. For different values of K, messages are exchanged between a different subset of the total P^2 processor pairs.

Figure 12 shows the estimated completion times for the heuristic and direct schedules, as calculated by the simulator. The lower bound on the completion time is also shown in these figures. It can be seen that the completion time of our heuristic algorithm is always within 2 - 10 % of the lower bound.

An important characteristic of our heuristic algorithm is that the schedule is adaptive to variations in network performance. In a metacomputing system, applications can query a directory service for current values of network performance. Our heuristic can use such information during the communication scheduling phase. In contrast, the direct algorithm uses a fixed communication pattern, irrespective of the network performance. Figure 13 shows the simulated communication time of both schedules as the heterogeneity in the network is varied. Here, $P = 64$, $K = 40$, and $N = 10000000$. The variation in network bandwidth is increased from 100 to 1900%. When the variation is 100 %, the network bandwidth varies in the range 10 MB/s to 20 MB/s. When the variation is 1900 %, the network bandwidth varies in the range of 10 MB/s to 200 MB/s. The completion time of our heuristic algorithm is always very close to the lower bound, for all values of network heterogeneity.

6. Conclusion

In this paper, we have developed an effective communication scheduling technique for the important problem of block-cyclic redistribution over a heterogeneous network. Our techniques target an emerging class of computational platforms, namely workstation clusters and distributed systems. The interconnection network in such systems is typically heterogeneous and shared. Our adaptive algorithms derive communication schedules using run-time information on network performance and heterogeneity.

Our scheduling techniques can lead to significant improvements in application performance. Our experimental results show reductions of upto 60 % in communication time, compared with widely used algorithms for homogeneous systems.

Several interesting directions for future research can be identified. The performance of our communication scheduling algorithms must be experimentally evaluated using a real heterogeneous network. Such experiments would also point to further enhancements and refinements to our communication model. Techniques to reduce the cost of the communication scheduling phase must also be investigated. Our communication scheduling technique can be easily extended to other important redistribution problems, such as block-cyclic redistribution of multi-dimensional arrays, and redistribution from *cyclic*(x) to *cyclic*(y).

Acknowledgments

Prashanth B. Bhat was supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931. Viktor K. Prasanna was partially supported by NSF under grant CCR-9317301.

References

[1] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling block-cyclic array redistribution. *IEEE Trans. Parallel and Distributed Systems*, 9(2):192–205, Feb. 1998.

[2] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. In *Proc. 5th Intnl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.

[3] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.

[4] S. D. Kaushik, C. H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase array redistribution: Modeling and evaluation. In *Proc. Intnl. Parallel Processing Symposium*, pages 441–445, Apr. 1995.

[5] J. Kim and D. J. Lilja. Exploiting multiple heterogeneous networks to reduce communication costs in parallel programs. In *Proc. Heterogeneous Computing Workshop*, pages 83–95, April 1997.

[6] J. Kim and D. J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intnl. Symp. High Performance Distributed Computing*, 1997.

[7] H. Korab and M. D. Brown, *Eds. Virtual Environments and Distributed Computing at SC '95: GII Testbed and HPC Challenge Applications on the I-WAY.* ACM/IEEE Supercomputing '95, 1995.

[8] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient algorithms for block-cyclic redistribution of arrays. *Algorithmica*, to appear.

[9] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient data remapping algorithms for embedded signal processing applications. In *Proc. 10th Intnl. Conference on High Performance Computers*, June 1996.

[10] N. Park, V. K. Prasanna, and C. S. Raghavendra. Efficient algorithms for block-cyclic array redistribution between processor sets. In *Proc. Supercomputing '98.*

[11] S. Ramaswamy and P. Banerjee. Automatic generation of efficient array redistribution routines for distributed memory multicomputers. In *Proc. 5th Symposium on Frontiers of Massively Parallel Computation*, pages 342–349, Feb. 1995.

[12] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Computing*, 23(3):617–632, June 1994.

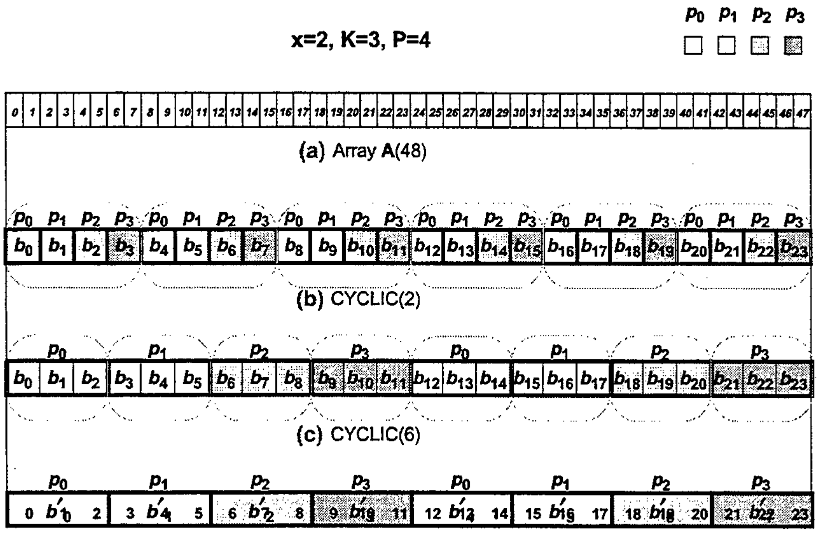
[13] L. Smarr and C. E. Catlett. Metacomputing. *Commns. of the ACM*, 35(6):45–52, June 1992.

[14] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, Mar. 1998.

[15] R. Thakur, A. Choudhary, and J. Ramanujam. Efficient algorithms for array redistribution. *IEEE Trans. Parallel and Distributed Systems*, 7(6):587–594, June 1996.

[16] D. W. Walker and S. W. Otto. Redistribution of block-cyclic data distributions using MPI. Technical Report ORNL/TM-12999, Oak Ridge National Labs, June 1995.

[17] L. C. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Wittig. Issues of reserving resources in advance. In *Proc. 5th Intnl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.


x=2, K=3, P=4
P_0 P_1 P_2 P_3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15} b_{16} b_{17} b_{18} b_{19} b_{20} b_{21} b_{22} b_{23}
P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15} b_{16} b_{17} b_{18} b_{19} b_{20} b_{21} b_{22} b_{23}
P_0 P_1 P_2 P_3 P_0 P_1 P_2 P_3
0 b'_0 2
3 b'_1 5
6 b'_2 8
9 b'_3 11
12 b'_4 14
15 b'_5 17
18 b'_6 20
21 b'_7 23

(a) Array **A**(48)

(b) CYCLIC(2)

(c) CYCLIC(6)

(d) CYCLIC(6)
with new block indices

Figure 1. Redistribution from *cyclic*(2) to *cyclic*(6) on 4 processors.

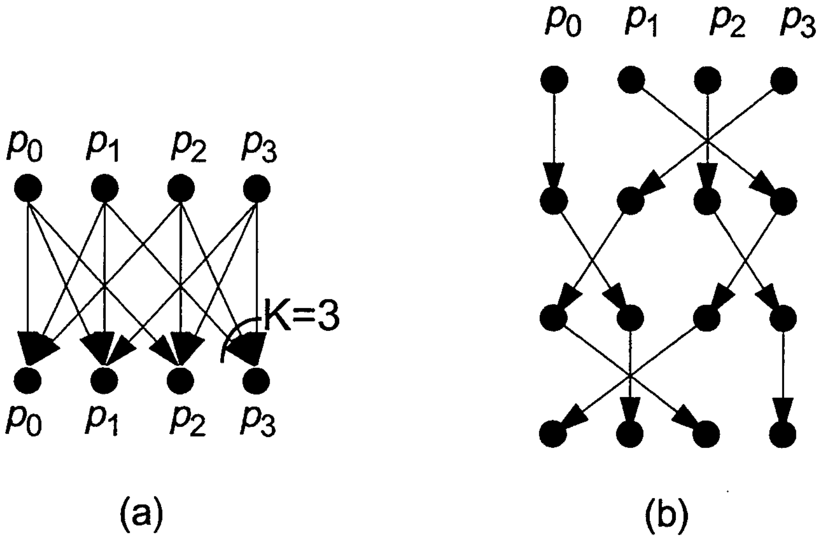


Figure 2. (a) Communication pattern for $\Re_x(3,4)$. (b) Contention-free schedule for a homogeneous network.

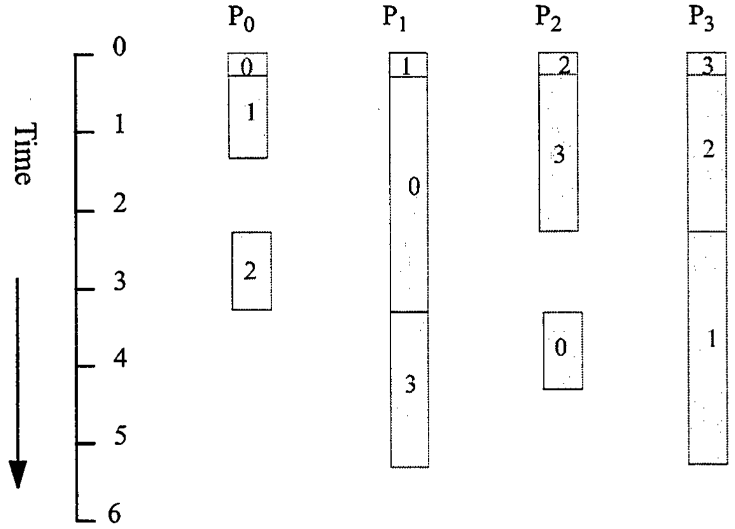


Figure 3. A communication schedule for $\Re_x(3,4)$.

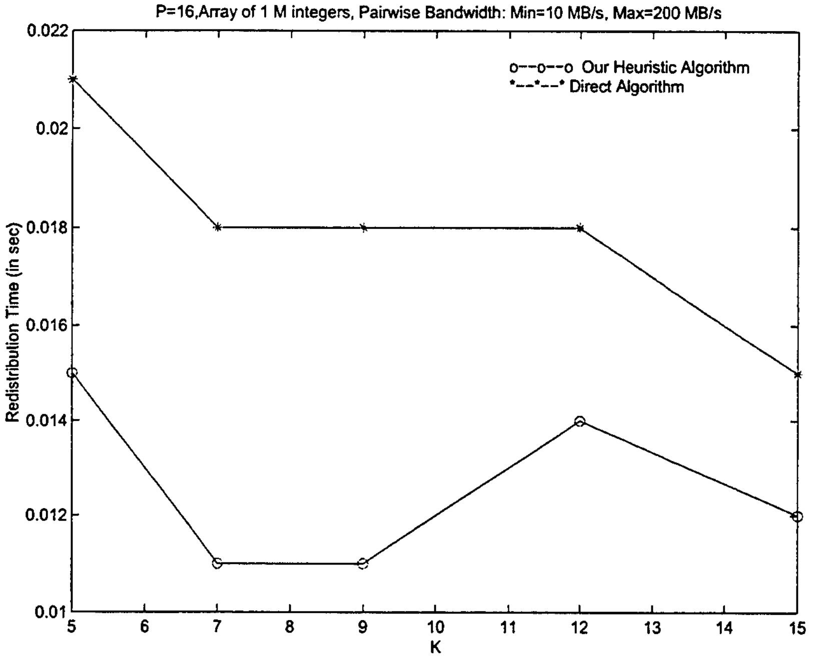


Figure 4. Completion times of our open shop heuristic schedule and the direct schedule on 16 processors of the Cray T3E.

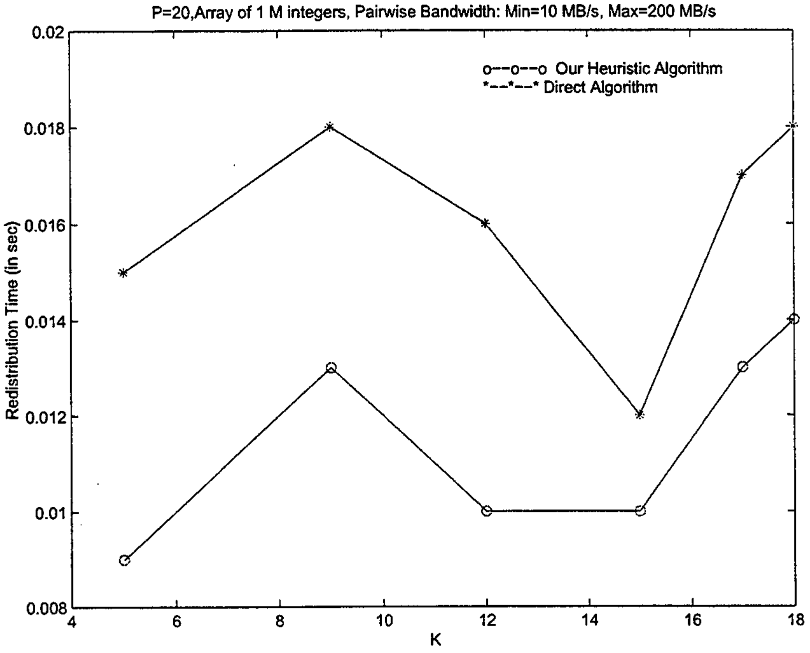


Figure 5. Completion times of our open shop heuristic schedule and the direct schedule on 20 processors of the Cray T3E.

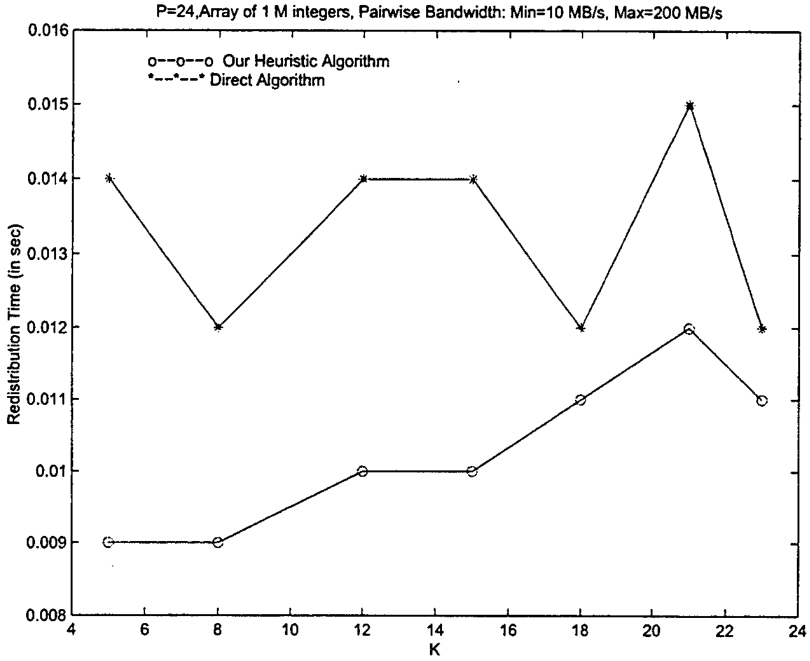


Figure 6. Completion times of our open shop heuristic schedule and the direct schedule on 24 processors of the Cray T3E.

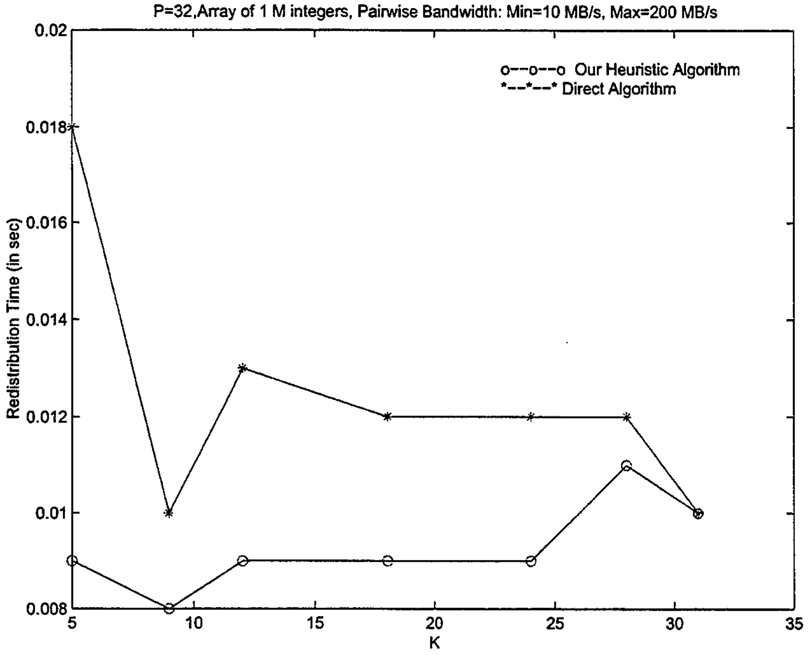


Figure 7. Completion times of our open shop heuristic schedule and the direct schedule on 32 processors of the Cray T3E.

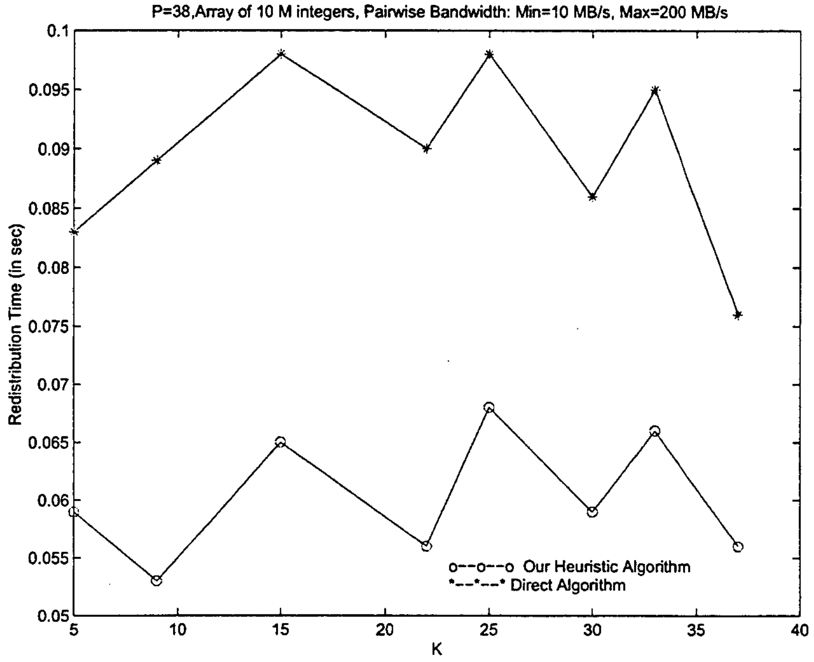


Figure 8. Completion times of our open shop heuristic schedule and the direct schedule on 38 processors of the Cray T3E.

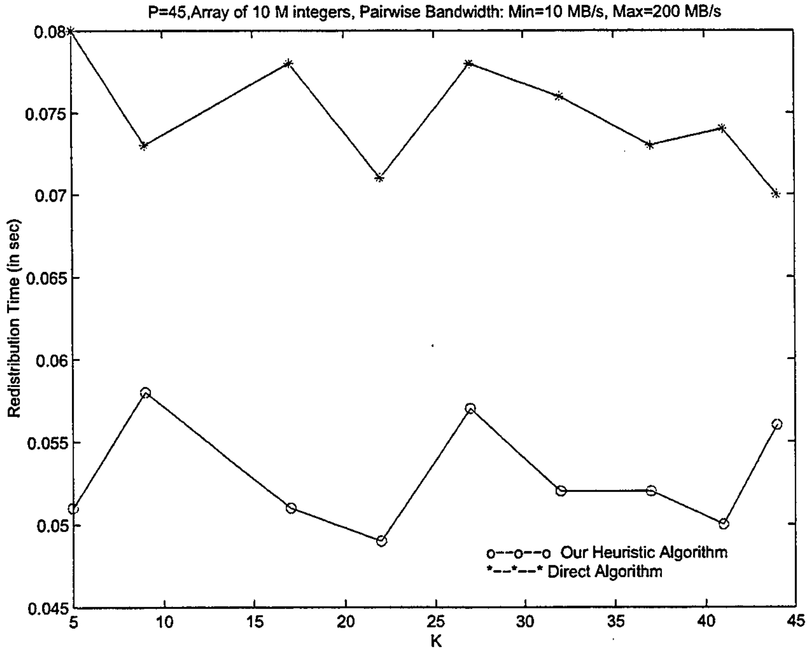


Figure 9. Completion times of our open shop heuristic schedule and the direct schedule on 45 processors of the Cray T3E.

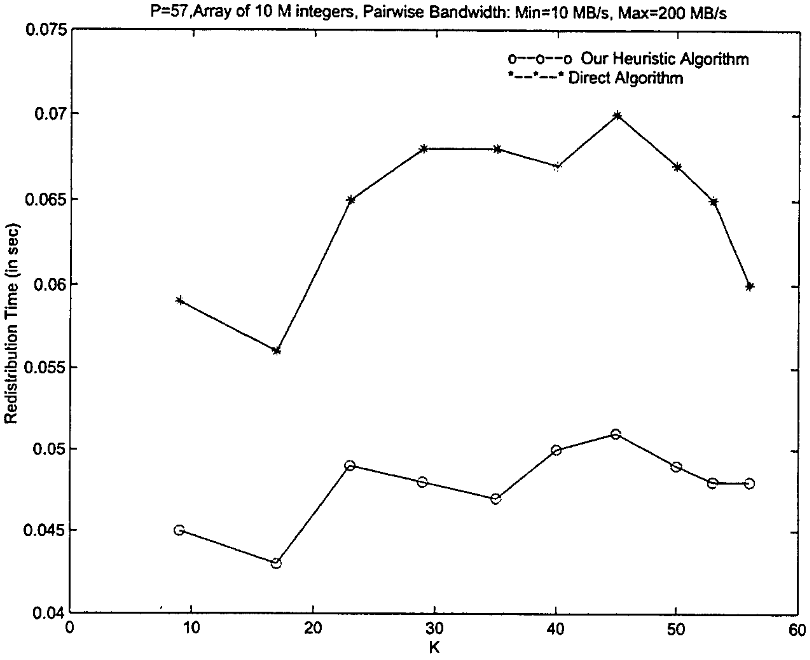


Figure 10. Completion times of our open shop heuristic schedule and the direct schedule on 57 processors of the Cray T3E.

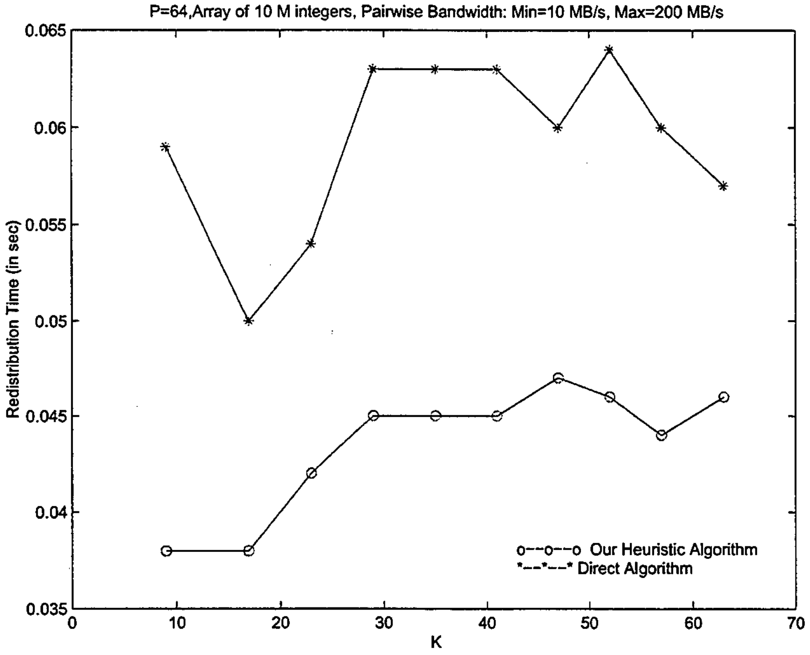


Figure 11. Completion times of our open shop heuristic schedule and the direct schedule on 64 processors of the Cray T3E.

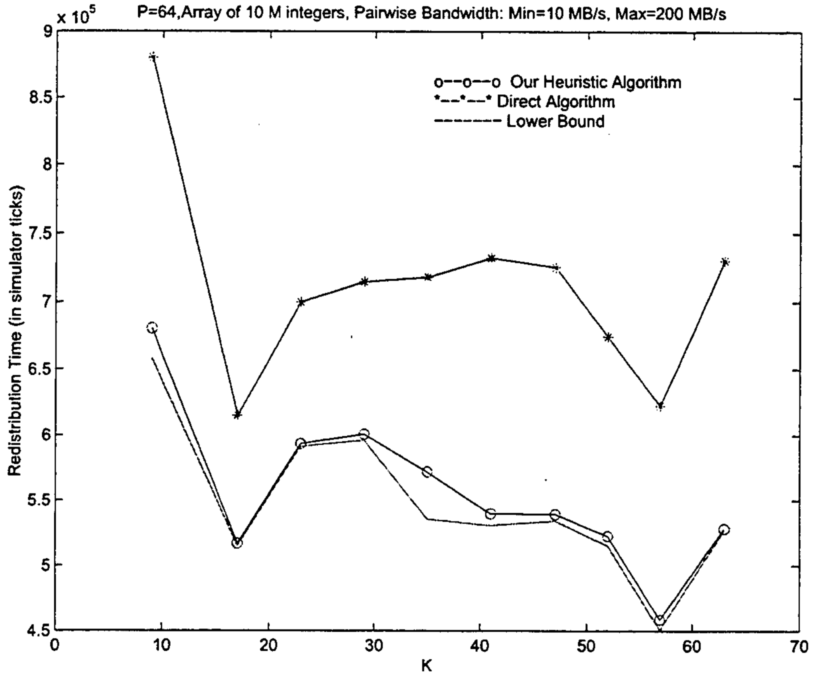


Figure 12. Simulation results: lower bound, heuristic schedule, and direct algorithm. P=64 processors, K=9 to 63.

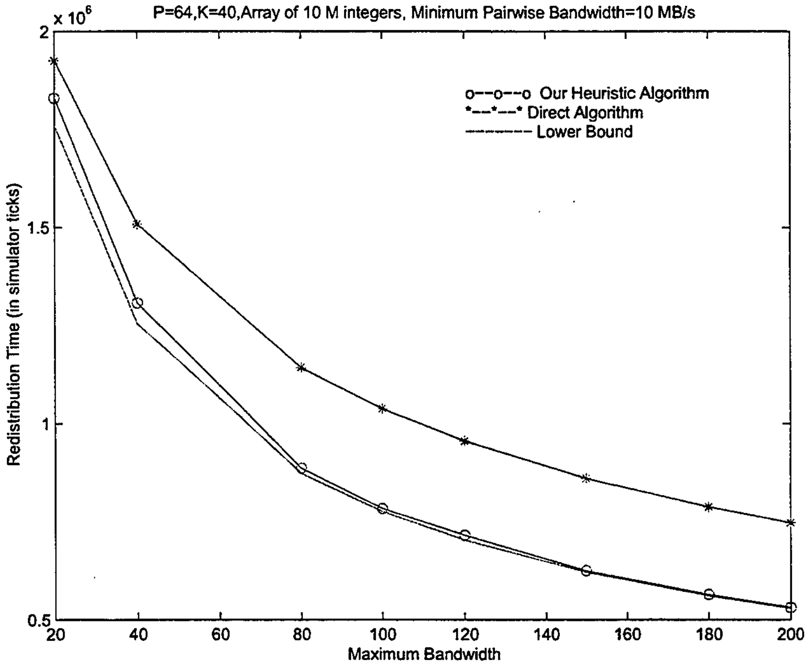


Figure 13. Simulation results: lower bound, heuristic schedule, and direct algorithm. P=64, K=40, network heterogeneity varies from 100% to 1900%.

Efficient Collective Communication in Distributed Heterogeneous Systems

Prashanth B. Bhat *
Dept. of EE-Systems, EEB 246
University of Southern California
Los Angeles, CA 90089-2562
prabhat@halcyon.usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
raghu@aero.org

Viktor K. Prasanna*
Dept. of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
prasanna@ganges.usc.edu

Abstract

The Information Power Grid (IPG) is emerging as an infrastructure that will enable distributed applications – such as video conferencing and distributed interactive simulation – to seamlessly integrate collections of heterogeneous workstations, multiprocessors, and mobile nodes, over heterogeneous wide-area networks. This paper introduces a framework for developing efficient collective communication schedules in such systems. Our framework consists of analytical models of the heterogeneous system, scheduling algorithms for the collective communication pattern, and performance evaluation mechanisms. We show that previous models, which considered node heterogeneity but ignored network heterogeneity, can lead to solutions which are worse than the optimal by an unbounded factor. We then introduce an enhanced communication model, and develop three heuristic algorithms for the broadcast and multicast patterns. The completion time of the schedule is chosen as the performance metric. The heuristic algorithms are FEF (Fastest Edge First), ECEF (Earliest Completing Edge First), and ECEF with look-ahead. For small system sizes, we find the optimal solution using exhaustive search. Our simulation experiments indicate that the performance of our heuristic algorithms is close to optimal. For performance evaluation of larger systems, we have also developed a simple lower bound on the completion time. Our heuristic algorithms achieve significant performance improvements over previous approaches.

1. Introduction

With recent advances in high-speed networks, distributed heterogeneous computing has emerged as an attractive computational paradigm. The Information Power Grid (IPG) [6] is emerging as an infrastructure that will connect distributed computational sites worldwide. This will create a universal source of computing power, thereby providing pervasive and inexpensive access to advanced computational capabilities. A typical grid-based distributed computing system will consist of a collection of heterogeneous workstations, multiprocessors, and mobile nodes. These nodes communicate with one another using a common set of protocols over different types of communication links, such as ATM, FDDI, Ethernet, and wireless channels. An example of such a system is shown in Figure 1. Such a distributed computing system is heterogeneous both in the computing nodes and in the communication network.

Several research projects, such as Globus [7], Legion [9], and MSHN [13] are developing toolkits and infrastructure support to enable the use of these systems for high performance computing. The issue of data dissemination middleware for wide-area network collaboratories is also being investigated [12]. Our research is a part of the MSHN project [13], which is a collaborative effort between DoD (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). MSHN (Management System for Heterogeneous Networks) is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. The goal is to schedule shared compute and network resources among individual applications so that their QoS requirements are satisfied.

The availability of high-speed wide-area networks has also enabled collaborative multimedia applications such as video conferencing, distributed interactive simulation, and collaborative visualization. For example, the *FACE* project [16] organized world-wide teleconferences among agents in Japan, USA, and the UK. The participating sites in these applications exchange large volumes of multimedia data, such as voice and video. Using the Internet, messages were propagated in about 60 *msec* between sites in Japan, while it took about 240 *msec* between Japan and Europe [16].

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

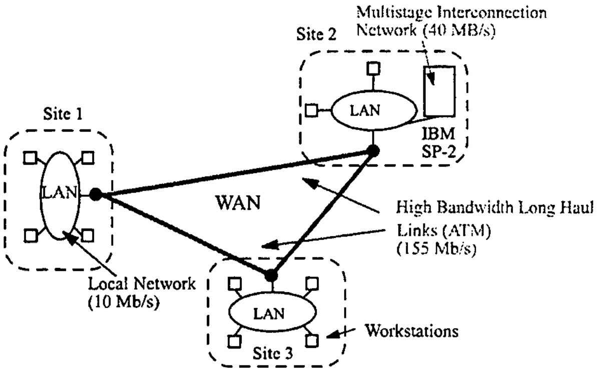

Multistage Interconnection Network (40 MB/s)
Site 2
LAN
IBM SP-2
Site 1
LAN
WAN
High Bandwidth Long Haul Links (ATM) (155 Mb/s)
Local Network (10 Mb/s)
LAN
Workstations
Site 3

Figure 1. A typical distributed heterogeneous system.

In both of the above scenarios, *viz.,* distributed high performance computing and collaborative multimedia applications, it is extremely important to efficiently perform group communication over a heterogeneous network. Typical group communication patterns are multicast, broadcast, and total exchange. In the multicast pattern, a source node sends the same message to a subset of nodes in the system. The broadcast pattern is a special case of multicast where the message is sent from a source to all the other nodes. In the total exchange communication pattern, every node sends a distinct message to every other node. The goal is to optimize a specified performance measure, *eg.,* minimize the time at which all the messages have been delivered.

In this paper, we develop efficient algorithms for broadcast and multicast in heterogeneous computing environments. These communication patterns occur in several military and commercial applications. In the battlefield, rapid dissemination of work orders and threat scenarios is critical [17]. A global satellite and ground-based networks are used in military battlefield to broadcast messages. The satellite sends the message to a group of base stations as it passes over them. The base stations then co-operatively broadcast the message to the other destinations over ground-based networks. The Internet can also be used to rapidly disseminate important emergency messages. In the past, broadcast and multicast problems have been studied extensively in the context of homogeneous and worm-hole routed networks [10, 14]. Similarly, multicast protocols such as CBT [2], DVMRP, and PIM [5] are now being deployed in wide-area networks. However, these techniques are not appropriate for the distributed network scenarios that we consider in this paper. For example, flooding is a technique where a node simultaneously sends the broadcast message to all its neighbors. The receiving nodes "flood" their neighbors in turn, until the message is received by all nodes. Some of the nodes could receive the message multiple times, depending on the network topology. Such techniques will not be efficient in wide-area heterogeneous networks, since each point-to-point communication event incurs an additional communication cost. Further, this will also introduce extra network congestion.

Recent research efforts [3] have investigated the problem of efficient broadcast and multicast in a network of heterogeneous workstations. The heterogeneity in the communication capabilities of the workstations was represented by associating a message initiation cost with each workstation. However, heterogeneity in the network was not considered. Based on this communication cost model, heuristic algorithms were developed for the broadcast and multicast problems. The heuristics achieve near-optimal performance for up to 10 nodes. In Section 2, we show that such a communication model can be very ineffective in a system with a heterogeneous network. We give examples where the completion time of a broadcast schedule using such a model is larger than the optimal completion time by an unbounded factor. It is therefore necessary to use a communication framework that considers heterogeneity in both the nodes and network links. Section 3 introduces our new communication model and framework. Our model represents the communication cost between two nodes P_i and P_j using two parameters: (i) a start-up time which accounts for the message initiation cost at P_i, and the network latency from P_i to P_j, and (ii) a data transmission cost which depends on the message size and the bandwidth from P_i to P_j. Using this model, we can consider the distributed system to be a fully connected network with a communication cost C_{ij} between every pair of nodes P_i and P_j. We do not assume a symmetric network, *i.e.,* $C_{ij} \neq C_{ji}$.

Since the problem of finding the optimal broadcast schedule in such a heterogeneous system is NP-complete, we have developed heuristic algorithms based on our communication framework. Our heuristic algorithms produce near optimal solutions for up to 10 nodes when tested with random networks. For larger size systems, it is extremely time consuming to compute the optimal solution. We have therefore developed a lower bound on the completion time. We evaluate the different heuristics by comparing their completion time with the lower bound.

The rest of the paper is organized as follows. In Section 2, we discuss related work and its shortcomings. Section 3 presents our formal model and general framework for collective communication in heterogeneous distributed computing environments. In Section 4, we present several heuristic algorithms for the broadcast and multicast problems. Section 5 compares the performance of our heuristics with previous algorithms, using simulation results. Section 6 identifies future research directions.

2. Shortcomings of Previous Research

Collective communication in homogeneous workstation networks and tightly coupled parallel systems has been thoroughly researched over the years. Communication libraries for frequently used patterns such as *total exchange, one-to-all broadcast, all-to-all broadcast,* and *gather* have been developed [1, 4, 18, 19].

However, collective communication in heterogeneous systems has not been investigated until very recently [11, 3]. The Efficient Collective Operations (ECO) [11] package was developed for networks of heterogeneous workstations. It implements the same functionality as the collective communication suite in the MPI standard. The ECO approach consists of first partitioning the network into subnets. A subnet consists of hosts which are in the same physical network. The collective communication then proceeds in two phases, inter-subnet and intra-subnet. However, such a two-phase strategy does not always ensure efficient implementations of collective communication patterns. This is especially true if the the inter-subnet links are much slower than the intra-subnet links.

Banikazemi *et. al.* [3] identified the important problem of performing efficient broadcast and multicast among a cluster of heterogeneous workstations. A homogeneous network was assumed. Their communication model associates a message initiation cost T_i with each of the N workstations. T_i is incurred whenever the i^{th} workstation (P_i) sends a message, independent of the identity of the receiving workstation. Based on this communication cost model, it was shown that broadcast schedules based on binomial trees, which achieve good results in homogeneous systems, can be very ineffective. A $N-1$ step heuristic algorithm, called Fastest Node First (FNF), was developed. Each step of the heuristic selects a sender and a receiver. The receiver is the node with the lowest T_i among the remaining receivers. The sender is the node that can complete the communication event at the earliest possible time. The FNF heuristic was evaluated for systems with up to 10 nodes [3]. For the examples considered, the completion time of the FNF heuristic was very close to the optimal.

However, there are scenarios where the performance of the FNF heuristic can be sub-optimal. Consider the example where the source has cost 1, there are n nodes with costs $n, n+1, n+2, \ldots, 2n-1$, and $2n$ slow nodes with very high costs. In the optimal schedule, the source would first send n messages to nodes with cost $2n-1, 2n-2, \ldots,$ respectively. At time n, the node with cost n has received the message from the source. Immediately after receiving the message, each of these nodes initiates a message to one of the slow nodes. During the time interval $[n, 2n]$, the source sends n more messages to the remaining slow nodes. The schedule completes at time $2n$.

In the FNF schedule, the source will send messages to nodes with cost $n, n+1, \ldots, 2n-1$ respectively. At time n, n nodes will have received the message. If each node immediately initiates a new message, each of the nodes with costs n to $\frac{3n}{2}$ can reach a slow node by time $2n$. During the time interval $[n, 2n]$, the source sends n more messages to n of the slow nodes. Thus, at time $2n$, $\frac{n}{2}$ of the slow nodes have not yet received the message. The schedule takes $\frac{n}{8}$ extra time units to complete. For large values of n, the completion time of the FNF schedule is much larger than the optimal.

A more significant shortcoming of [3] was the assumption of the homogeneous network. In a typical heterogeneous system, the communication cost depends both on the communication capability of the workstations as well as the network performance. Our paper investigates the impact of heterogeneity in both these aspects. We first illustrate the importance of considering network heterogeneity, using an example. Consider a system with 3 nodes, and pairwise communication costs as shown in Eq (1). The $(i, j)^{th}$ entry of $\mathbf{C}$ $(0 \leq i, j < 3)$ denotes the time to send the broadcast message from node P_i to P_j. This includes the message initiation cost on node P_i and also the network latency from P_i to P_j. Section 3 discusses this communication model in detail. Node P_0 is the source.

$$\mathbf{C} = \begin{bmatrix} 0 & 10 & 995 \\ 2000 & 0 & 10 \\ 70 & 5 & 0 \end{bmatrix} \qquad (1)$$

To develop a communication schedule based on node heterogeneity alone, we associate a communication cost T_i with each node. This is calculated as the average send cost from node P_i to all the other nodes. Thus, in Eq (1), $T_0 = 335, T_1 = 670, T_2 = 25$. We can now use the FNF heuristic [3] for this problem. Since the heuristic operates on a modified version of the input data (*i.e.,* the average communication costs), we call this the modified FNF heuristic.

For the example of Eq (1), the heuristic begins with node P_0 as the only sender. In the first step, P_2 is selected as the receiver. The communication from P_0 to P_2 takes 995 time units. Both these nodes are ready to send the next message at time 995. In the next step, node P_2 is selected as the sender and P_1 is selected as the receiver. This communication event takes 5 time units. The broadcast therefore takes 1000 time units to complete. Figure 2(a) shows this communication schedule.

However, it is easy to see that the optimal schedule takes only 20 time units. In the first step, P_0 sends a message to P_1 in 10 time units. In the next step, P_1 sends a message to P_2 in 10 time units. This schedule is shown in Figure 2(b). Thus, the use of a single message initiation cost for each node results in a communication schedule which is 50 times worse than the optimal schedule for this example. In the above example, we used the average send cost

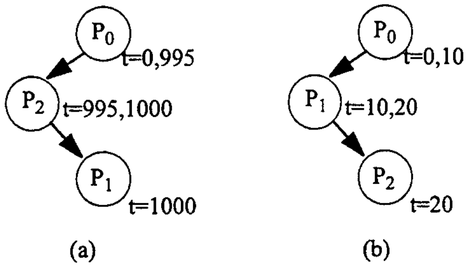


Figure 2. Broadcast schedules for the example in Eq (1): (a) Modified FNF schedule (b) Optimal schedule.

from each sender as its communication cost. Alternatively, we could have used the minimum send cost of each sender as its communication cost T_i. In Eq (1), the costs would then be $T_0 = 10, T_1 = 10, T_2 = 5$. It can be easily verified that the modified FNF heuristic again takes 1000 time units to complete.

The performance of the modified FNF heuristic would be still worse if the value of $C_{2,0}$ was larger. For example, if $C_{2,0}$ was 9995 instead of 995, the completion time would have been 10000 time units, *i.e.* 500 times the optimal completion time. We summarize this observation in the following lemma.

Lemma 1: In the presence of a heterogeneous network, there exist input instances for which the ratio of the completion time of the modified FNF heuristic to the optimal completion time is unbounded. □

Thus, communication models which consider only node heterogeneity can result in arbitrarily bad performance. It is therefore important to consider both node heterogeneity and network heterogeneity when designing communication algorithms for the broadcast problem.

3. A Communication Framework for Distributed Heterogeneous Systems

We now present our communication scheduling framework for distributed heterogeneous systems. The framework consists of three main components: (a) A communication model, (b) Scheduling heuristics, and (c) Performance metrics. In this section, we describe an enhanced communication model which incorporates node and network heterogeneity. Section 4 describes our heuristic algorithms for broadcast and multicast based on this model. The performance metric used in this paper is the completion time. Other candidate metrics are discussed in Section 7.

3.1. Communication Model

Consider a distributed heterogeneous system (Figure 1) with N nodes. We represent the computing nodes and network links in such a system using a directed graph G with N vertices. An edge (v_i, v_j) in G represents the path between nodes P_i and P_j, which could include links from multiple networks of different latencies and bandwidths. The weight C_{ij} of edge (v_i, v_j), $(0 \leq i, j < N)$ represents the time to send the broadcast message from P_i to P_j. If there exists at least one path between every pair of nodes in the system, G will be a complete graph. The graph is not necessarily symmetric, *i.e.* $C_{ij} \neq C_{ji}$, in general. The information can also be represented as a $N \times N$ communication matrix **C**, with entries C_{ij}, as shown in Eq (1).

Our communication model represents the network performance between any processor pair (P_i, P_j) using two parameters: a start-up cost T_{ij} and a data transmission rate B_{ij}. The time for sending a m byte message between these nodes is given by $T_{ij} + \frac{m}{B_{ij}}$. A similar communication model has been widely used for tightly-coupled homogeneous distributed memory systems with good results [20]. In networked heterogeneous systems, typical values for the start-up cost could be in the range of 10 to 500 μs, while typical values for the bandwidth could be in the range of kb/s to hundreds of Mb/s. Note that the communication time depends on the identities of both the sender and receiver, unlike previous models [3]. The model thus enables a realistic estimate of the communication time between any pair of nodes.

Table 1 is an example of measured network performance on the GUSTO testbed of the Globus distributed heterogeneous system [7]. The table shows four of the GUSTO sites: NASA AMES, Argonne National Lab, University of Indiana, and USC-ISI. Observe that the network performance varies considerably between different pairs of nodes, and depends on both the source and destination. For instance, the bandwidth between USC-ISI and AMES is much larger than the bandwidth between USC-ISI and IND. Previous communication models [3], which assume that the communication time from node P_i to node P_j is independent of P_j and depends only on the source node P_i, are therefore unlikely to be effective for such systems.

Our model assumes that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple messages to be transmitted simultaneously. Software support for nonblocking and multithreaded communication sometimes allows applications to initiate multiple send and receive operations. However, all these operations are eventually serialized by the single hardware port to the network. Our model

	AMES	ANL	IND	USC-ISI
AMES		34.5/512	89.5/246	12/2044
ANL	34.5/512		20/491	26.5/693
IND	89.5/246	20/491		42.5/311
USC-ISI	12/2044	26.5/693	42.5/311	

Table 1. Latency(ms) / Bandwidth(kbits/s) between 4 GUSTO sites.

accurately represents this phenomenon.

If multiple nodes simultaneously send to any node P_j, we say that node contention occurs at P_j. The model assumes that these messages are received one after the other at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation.

Based on the network performance parameters and our communication model, we can calculate the communication time to send the broadcast message between any pair of nodes in the heterogeneous network. This information is used to determine the edge weights of G and the entries of the communication matrix **C**. The communication matrix for broadcasting a 10 MByte message over the network of Table 1 is shown in Eq (2). Entries are in *sec.*

$$\mathbf{C} = \begin{bmatrix} 0 & 156 & 325 & 39 \\ 156 & 0 & 163 & 115 \\ 325 & 163 & 0 & 257 \\ 39 & 115 & 257 & 0 \end{bmatrix} \qquad (2)$$

4. Heuristics for Broadcast and Multicast

Consider a communication cost matrix **C** with N nodes. We first define a lower bound on any communication schedule for the broadcast and multicast problems, and then discuss our heuristic algorithms.

4.1. A Lower Bound

Let P_0 be the source of the broadcast or multicast operation, and **D** represent the set of destination nodes. $\mathbf{D} \subset \{P_1, P_2, \ldots, P_{N-1}\}$ for multicast, while $\mathbf{D} = \{P_1, P_2, \ldots, P_{N-1}\}$ for broadcast. For each node P_i in **D**, we can compute the shortest path from the source node P_0 to P_i. The weight of this path represents the earliest time at which the broadcast message from P_0 can reach P_i. This is therefore called the *Earliest Reach Time* of node P_i, denoted as ERT_i.

Lemma 2: A lower bound on any communication schedule for the broadcast or multicast problem is given by

$$LB = \max_{P_i \in \mathbf{D}} ERT_i \qquad (3)$$

Proof: We know that ERT_i represents the earliest time at which node P_i can be reached. From the definition of the broadcast and multicast communication pattern, the message must reach every node in **D**. Hence, no communication schedule can complete until the node with the maximum ERT is reached. Eq (3) therefore gives a lower bound on the completion time. □

The lower bound is not tight, since it assumes that the messages from the source to each destination can proceed in parallel. Thus, the optimal completion time could be significantly larger than the lower bound.

Lemma 3: For any instance of the multicast or broadcast problem, the optimal completion time is bounded by $\mid \mathbf{D} \mid \times LB$, *i.e.,*

$$\frac{OptimalCompletionTime}{LB} \leq \mid \mathbf{D} \mid \qquad (4)$$

Further, this ratio is tight.

Proof: The lower bound LB of Eq (3) is the communication time to send the message from the source to the farthest node. Thus, the communication time to send a message from the source to any node is $\leq LB$. We can always construct a communication schedule in which the source sequentially sends $\mid \mathbf{D} \mid$ messages to all the destinations. The $\mid \mathbf{D} \mid$ communication steps can therefore be completed in atmost $LB \times \mid \mathbf{D} \mid$ time units.

To prove that the ratio is tight, consider the broadcast problem on the communication cost matrix of Eq (5). In this matrix, $\mathbf{C}_{0j} = 10, (0 < j < N)$. Also, $\mathbf{C}_{ij} = 100, (0 < i < N, i \neq j)$. The diagonal entries $\mathbf{C}_{ii} = 0, (0 \leq i < N)$. The shortest path to every node P_i is the direct path (P_0, P_i). The lower bound would be the maximum outgoing edge from P_0, *i.e.,* 10. However, the optimal schedule has a completion time of $10 \mid \mathbf{D} \mid$. Thus, there exist examples wherein the optimal completion time is $\mid \mathbf{D} \mid$ times as large as our simple lower bound. □

$$\mathbf{C} = \begin{bmatrix} 0 & 10 & 10 & \ldots & 10 \\ 100 & 0 & 100 & \ldots & 100 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 100 & 100 & 100 & \ldots & 0 \end{bmatrix} \qquad (5)$$

4.2. Computing the Optimal Schedule

The possible number of communication schedules for a broadcast or multicast problem instance with N nodes is exponential in N. The completion times of these schedules can vary considerably, depending on the performance of the heterogeneous network links. Finding the optimal communication schedule is an NP-Complete problem. However, for systems with a small number of nodes, we can find the optimal schedule using exhaustive search. Our algorithm, which uses a branch-and-bound strategy, computes the optimal solution for up to 10 nodes in a reasonable amount of time. For small system sizes, we shall compare the performance of our heuristic algorithms with the optimal solution.

4.3. Our Heuristic Algorithms

Our algorithms for the broadcast and multicast problems can be described using the following formalism. The nodes are partitioned into three sets, **A**, **B**, and **I**. At any time, set **A** consists of nodes which have already received the message. Set **B** consists of nodes which must receive the message in the future. **I** contains the other nodes. Initially, set **A** consists of the source node while set **B** consists of the destination nodes for the multicast, *i.e.* **B**=**D**. For the broadcast problem, $\mathbf{I} = \phi$.

At every step, a sender from **A** and a receiver from **B** are chosen. For the multicast problem, the message could also be relayed through one of the nodes in **I**, if this path incurs lower communication time. After each communication event, the receiver node (and the intermediate node, if one was chosen) is moved to **A**. The communication schedule involves $\mid \mathbf{D} \mid$ such steps. We now present the baseline algorithm and our FEF, ECEF, and look-ahead heuristics.

Baseline Algorithm

We use the modified FNF heuristic [3] as a baseline algorithm. This algorithm associates a single communication cost with each node rather than a distinct cost for each pair of nodes. We use the average send cost from node P_i to all the other nodes as its communication cost T_i. The FNF heuristic algorithm [3] consists of $N-1$ steps. At every communication step, the node from **B** with the lowest T_j is chosen as the receiver. A sender is chosen such that the communication event can be completed at the earliest possible time. This is the node P_i which has the minimum value of

$$R_i + T_i \qquad (6)$$

where R_i is the ready time of the sender P_i.

Fastest Edge First (FEF):

Each step of our FEF heuristic selects the smallest weight edge (i, j) where P_i belongs to **A** and P_j belongs to **B**. The choice of the edge determines both the sender and receiver node for the corresponding communication step. P_j is then moved from **B** to **A**. The communication step starts at R_i, and takes $\mathbf{C}_{i,j}$ time units. During this time, both P_i and P_j are busy.

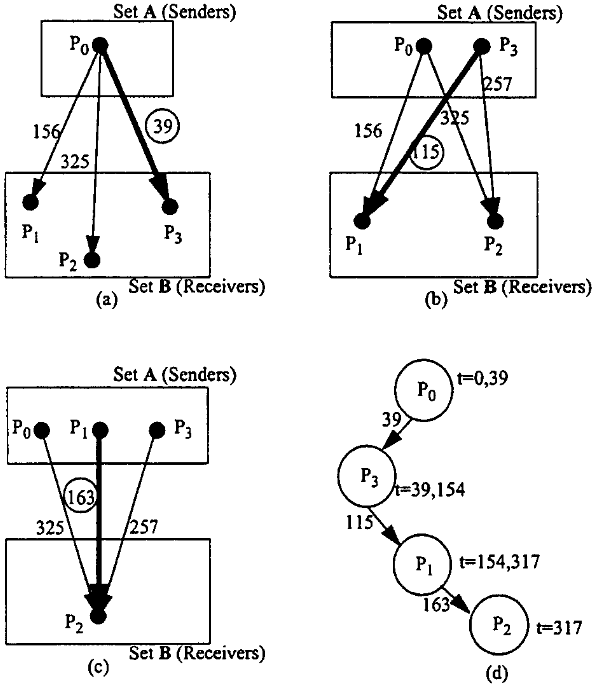


Figure 3. FEF communication schedule for the 4 node example of Eq (2).

The algorithm initially sorts the outgoing edges from each node in increasing order of their weights. This phase takes $O(N^2 \log N)$ time. The senders in **A** are then sorted in increasing order of their minimum weight outgoing edge. The new node added to set **A** at every step is inserted into the sorted sender list based on its minimum weight outgoing edge. The algorithm terminates after all the destination nodes have been moved to **A**. This involves $N-1$ steps for the broadcast algorithm and a maximum of $N-1$ steps for the multicast algorithm. The running time for this phase is also $O(N^2 \log N)$. The overall running time of the FEF heuristic is therefore $O(N^2 \log N)$.

Figure 3 shows the steps in the FEF heuristic for the broadcast problem in the 4 node system of Eq (2). Figure 3(a) shows the initial situation when set **A** contains only the source node, and set **B** contains the other nodes. The figures show the edge weights of only the edges in the **A-B** cut. Figures 3(b)-3(c) show the sequence in which the FEF heuristic moves edges from **B** to **A**. Figure 3(d) shows the broadcast tree for this schedule.

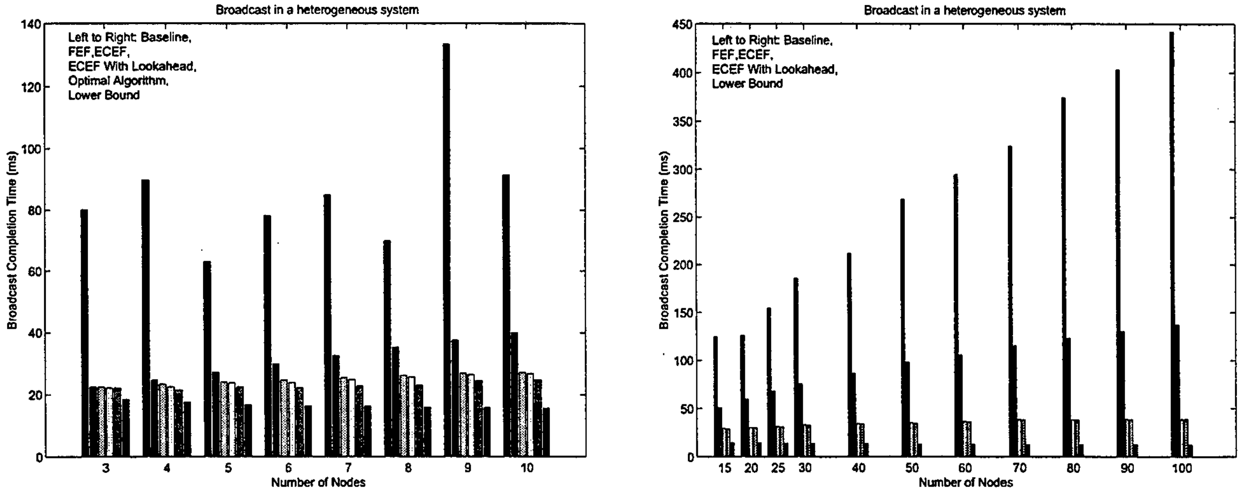


Figure 4. Simulation results for broadcast in a heterogeneous system.

Earliest Completing Edge First (ECEF):

The structure of our ECEF heuristic algorithm is similar to the FEF heuristic. At every step, an edge (i, j) is selected, where P_i belongs to **A** and P_j belongs to **B**. The choice of the edge considers both the weight of the edge and the ready time of the sender. The chosen communication event is the one that can complete earliest. Thus, the chosen edge is the one that minimizes the sum

$$R_i + \mathrm{C}_{i,j} \tag{7}$$

over all senders P_i and receivers P_j, where R_i is the ready time of sender P_i. As in the FEF heuristic, a sorted list of senders is maintained. The senders are sorted based on both their ready time and their minimum weight outgoing edge. The heuristic has a running time of $O(N^2 \log N)$.

Look-ahead Algorithm:

Our look-ahead algorithm is an enhanced version of the ECEF heuristic. At each step of the heuristic, a *look-ahead value* L_j is calculated for each node P_j in **B**. This value quantifies the "goodness" of moving node P_j from **B** to **A**. At each step, the algorithm first computes the value of L_j for all nodes in **B**. As in the ECEF heuristic, an edge is then selected from the **A-B** cut. The chosen edge is the one that minimizes the sum

$$R_i + \mathrm{C}_{i,j} + L_j \tag{8}$$

The look-ahead function can be defined in several ways. We have used the following look-ahead measure.

$$L_j = \min_{P_k \in \mathbf{B}} \mathrm{C}_{j,k} \tag{9}$$

Thus, for a given node P_j in **B**, the minimum communication cost from itself to all the other nodes in **B** is used as the look-ahead value. Intuitively, such a look-ahead function increases the usefulness of P_j as a sender, if it is moved to **A**.

The running time of the look-ahead algorithm is $O(N^3)$, since the evaluation of the look-ahead measure for each element of **B** at every step takes $O(N)$. Alternative look-ahead functions can also be used, such as the average of the communication costs from P_j to other nodes in **B**. L_j could also be calculated as the average cost of senders to receivers, assuming that P_j is made a sender. This look-ahead function has a computational complexity of $O(N^2)$, and the overall running time will therefore be $O(N^4)$. Our experiments in Section 5 use the look-ahead measure of Eq (9).

5. Experimental Results

We have developed a software simulator that executes the heuristic algorithms of Section 4, and calculates the completion time for each of them. The inputs to the simulator are the number of nodes, the size of the message to be broadcast or multicast, and the range of start-up times and bandwidths in the heterogeneous network. The simulator generates a random communication matrix based on these parameters. For the case of multicast, the number of destinations is given as input, and the simulator randomly chooses destination nodes. The simulator then executes the heuristic algorithms on 1000 random input configurations and reports the average completion times.

Figure 4 compares the performance of the different communication scheduling heuristics for the broadcast problem with a message size of 1 MB. The pairwise network laten-

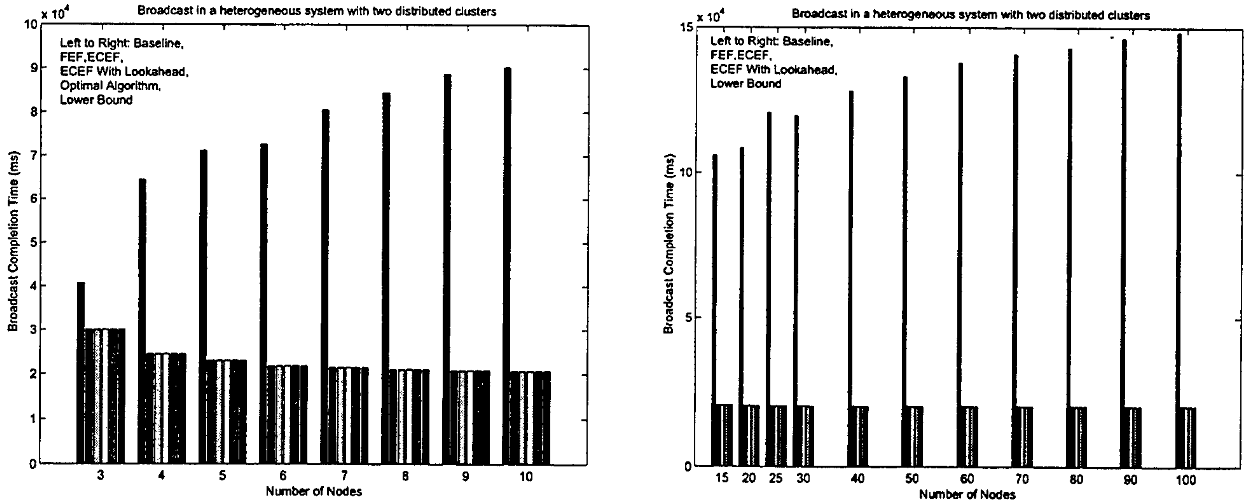


Figure 5. Simulation results for broadcast in a heterogeneous system with 2 distributed clusters.

cies and bandwidths are chosen in the ranges of 10 μsec to 1 $msec$, and $10kB/s$ to $200MB/s$ respectively. The graph shows the completion time for the baseline algorithm, the FEF, ECEF, and look-ahead heuristics, and our simple lower bound. For small system sizes (upto 10 nodes), the optimal completion time is also shown. Since our lower bound is not tight, it is typically much lesser than the optimal completion time. The graph shows that the completion time of our heuristic algorithms is always close to the optimal. The ECEF and look-ahead algorithms have a lower completion time than that of the FEF heuristic. The completion time of the baseline algorithm is significantly larger than that of the other heuristics. This shows the benefit of using a communication model which accurately represents heterogeneity in the network, as well as in the nodes.

The performance advantage of our heuristic algorithms over the baseline algorithm can also be seen in Figure 5. Figure 5 considers a system with two distinct geographically distributed clusters. It is assumed that half the nodes are in the first cluster, while the other nodes are in the second cluster. The heterogeneous network is assumed to be fast within each cluster, but is slow across clusters. For the intra-cluster networks, the latencies and bandwidths are in the ranges of 10 μsec to 1 $msec$, and $10MB/s$ to $200MB/s$ respectively. For the inter-cluster networks, the latencies and bandwidths are in the ranges of $1msec$ to $20msec$, and $10kB/s$ to $50kB/s$ respectively. As before, the size of the broadcast message is 1 MB.

Figure 6 shows the completion time for multicast in a 100 node system. The number of multicast destinations is increased from 5 to 90. For the case of k destinations, 1000 experiments are performed with k randomly chosen destinations. The average completion time is plotted in Figure 6.

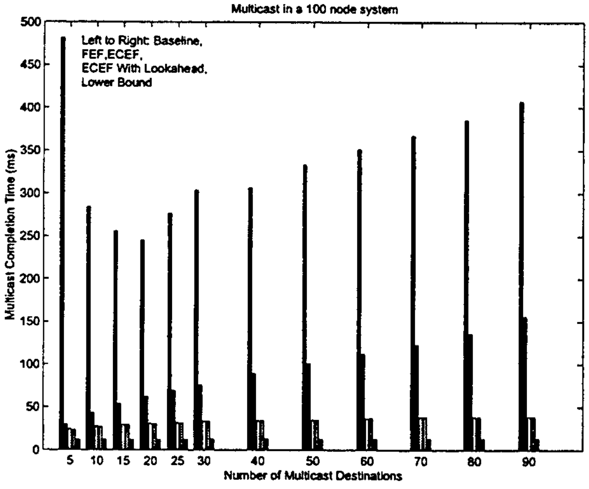


Figure 6. Simulation results for multicast.

Observe that the heuristic algorithms again significantly outperform the baseline algorithm.

6. Research Issues

The experimental results of Section 5 clearly show the performance benefits of our heuristic algorithms. However, there are scenarios in which some of our heuristics can have poor performance. Consider the asymmetric communication cost matrix of Eq (10), which could be a system with Asymmetric Digital Subscriber Lines (ADSL).

$$\mathbf{C} = \begin{bmatrix} 0 & 2 & 2 & 2.1 & 2 \\ 100 & 0 & 100 & 100 & 100 \\ 100 & 100 & 0 & 100 & 100 \\ 1000 & 0.1 & 0.1 & 0 & 0.1 \\ 100 & 100 & 100 & 100 & 0 \end{bmatrix} \qquad (10)$$

In the optimal broadcast schedule, P_0 sends the message to P_3 in step 1, and then P_3 then sends messages to the other nodes in steps 2, 3, and 4. This has a completion time of 2.4 time units. However, the ECEF heuristic sends the message from P_0 to P_1 in step 1,P_0 to P_2 in step 2, P_0 to P_4 in step 3, and P_0 to P_3 in step 4. The completion time is 8.4 time units. The look-ahead algorithm does find the optimal schedule. It chooses the node P_3 as the receiver in the first step, since P_3 has a low-cost outgoing edge.

However, the performance of the look-ahead schedule is poor for the communication matrix of Eq (11). The algorithm takes 4.1 time units (P_0 to P_1, P_1 to P_2, P_2 to P_3, and P_3 to P_4). The optimal schedule takes only $2.2 + 2\epsilon$ time units (P_0 to P_3, P_3 to P_4, P_4 to P_1, P_4 to P_2). For larger systems, the difference between the completion times of the look-ahead and optimal schedules can be much higher.

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 1 & 1.1 & 100 \\ 100 & 0 & 1 & 100 & 100 \\ 100 & 100 & 0 & 1 & 100 \\ 100 & 100 & 100 & 0 & 1.1 \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 \end{bmatrix} \qquad (11)$$

However, communication matrices such as Eq (11) do not typically occur in real scenarios. Often, **C** is symmetric. The triangle inequality is also usually valid, *i.e.*,

$$\mathbf{C}_{ij} \leq \mathbf{C}_{ik} + \mathbf{C}_{kj}, 0 \leq k < N \qquad (12)$$

For such a system, stronger performance bounds than Eq (4) could be shown. We are investigating this issue.

We are also investigating new heuristic schedules based on the Minimum Spanning Tree(MST) and Steiner Tree algorithms. The steps in our FEF algorithm are identical to Prim's MST algorithm. We are currently investigating a *progressive MST* approach. This is an enhancement to Prim's algorithm which accounts for the ready time of each node. After each step of the algorithm, some of the edge weights are updated to reflect the change in ready times. We are also investigating a *two-phase* approach. During the first phase, a MST is constructed. The structure of the MST is used to guide the selection of intermediate nodes for the second phase, which constructs the heuristic schedule.

The main difference between the MST problem and our broadcast problem is the cost metric. The metric in the MST problems is usually the total weight of edges in the spanning tree. In contrast, the completion time of the broadcast and multicast problems is the time at which all nodes have received the message. Delay-constrained MST problems, which minimize the maximum delay between the source and any destination, have also been considered [15]. However, this metric is also different from the completion time. Consider the example of Eq (10). The delay-constrained algorithm would create a MST with edges $(P_0, P_1), (P_0, P_2), (P_0, P_3)$, and (P_0, P_4). Although the maximum delay is 2.1, the completion time is 8.1 time units. In fact, if the triangle inequality of Eq (12) holds, the delay-constrained algorithm will always send $\mid \mathbf{D} \mid$ messages sequentially from the source to each destination.

A second difference is that the widely known MST algorithms of Prim and Kruskal were developed for undirected graphs. Our progressive and two-phase techniques can build upon these techniques if the heterogeneous network is symmetric. For asymmetric networks, MST algorithms for directed graphs can be used [8].

In designing a heuristic, we must give special attention to two kinds of nodes: (a) Nodes which are hard to reach from every other node, and are also unable to reach other nodes quickly. The message to such a node should be sent early in the schedule, so that this communication event does not delay the completion time. (b) Nodes which are a little hard to reach, but which can reach many other nodes very easily. Such nodes should be selected early, so that they can relay the message to the other nodes.

We are therefore exploring an alternating *near-far* approach. All nodes are initially sorted in increasing order of their ERT. In the first two steps, messages are sent to the nearest node (say P_i), and to the farthest node (say P_j). From this point onwards, P_i and its recipients will send messages to the *near* nodes. This group always selects the nearest unreached node at every step. P_j and its recipients will send messages to the *far* nodes. This group selects the farthest unreached node. Such a near-far strategy is likely to balance the two conflicting goals discussed above.

For the multicast problem, we shall enhance our algorithm to relay messages through nodes in the intermediate set **I**, defined in Section 4.3. Our current algorithm does not incorporate this aspect. The problem of scheduling multiple simultaneous multicasts will also be considered.

The previous sections have illustrated the use of our framework for a specific cost model and performance metric. We now discuss some variations and extensions of these components. Our communication model assumed that a node can send and receive atmost one message at any time. In a *non-blocking* communication model, this assumption is relaxed. After an initial start-up time, the sender can initiate a new message. The first message is completed by the network without further intervention by the sender. Thus, a node could send out several messages before the first message reaches the receiver. Similar assumptions can be made at the receiver too.

We have used the completion time as our performance metric. *Robustness* metrics can be used to measure the ability of a communication schedule to reach all destinations, inspite of intermediate node or link failures. A communication schedule could increase its robustness measure by sending redundant messages for fault tolerance. Alternatively, acknowledgement schemes and time-out parameters could be used to detect failures before resending a message over a different path. Another candidate metric is the *amount of transmitted data*.

7. Conclusion

Efficient communication support is extremely important for several distributed computing scenarios, such as collaborative multimedia applications and parallel high performance computing over the IPG. This paper has introduced an analytical framework for designing efficient collective communication algorithms. The main components of our framework are a communication model to represent the heterogeneous network and nodes, performance metrics, and scheduling algorithms. Based on this framework, we have developed efficient solutions for broadcast and multicast. We have also identified several promising research directions to extend our work. We believe that future work along these directions can accelerate the widespread use of distributed heterogeneous computing.

Acknowledgment

We thank Mr. M. Banikazemi and Dr. D.K. Panda of the Ohio State University for providing us access to their program for computing the optimal communication cost in a system with heterogeneous nodes. Our branch-and-bound program to compute the optimal schedule in a system with a heterogeneous network uses some of their subroutines.

References

[1] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *IEEE Trans. Parallel and Distributed Systems*, 6(2):154–164, February 1995.

[2] T. Ballardie, P. Francis, and J. Crowcraft. Core Based Trees (CBT) – an architecture for scalable inter-domain multicast routing. In *Proc. ACM SIGCOMM*, pages 85 – 95, 1993.

[3] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proc. Intl. Conf. Parallel Processing*, pages 460–467, 1998.

[4] J. Bruck, D. Dolev, C.-T. Ho, M.-C. Rosu, and R. Strong. Efficient Message Passing Interface (MPI) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, 40(1):19–34, January 1997.

[5] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. G. Liu, and L. Wei. An architecture for wide-area multicast routing. In *Proc. ACM SIGCOMM*, 1994.

[6] I. Foster and C. Kesselman, *Eds. The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[8] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding Minimum Spanning Trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[9] A. S. Grimshaw and W. A. Wulf. Legion – a view from 50,000 feet. In *Proc. Fifth IEEE Intl. Symp. on High Performance Distributed Computing*, August 1996.

[10] X. Lin, P. K. McKinley, and L. M. Ni. Performance evaluation of multicast wormhole routing in 2D-mesh multicomputers. In *Proc. Intl. Conf. Parallel Processing, Vol. I*, pages 435–442, August 1991.

[11] B. B. Lowekamp and A. Beguelin. ECO: Efficient Collective Operations for communication on heterogeneous networks. In *Proc. 10th Intl. Parallel Processing Symposium*, pages 399–405, April 1996.

[12] G. R. Malan, F. Jahanian, and P. Knoop. Comparison of two middleware data dissemination services in a wide-area distributed system. In *Proc. Intl. Conf. Distributed Computing Systems*, May 1997.

[13] MSHN Web Page. *http://www.mshn.org*.

[14] D. K. Panda. Issues in designing efficient and practical algorithms for collective communication on wormhole-routed systems. In *ICPP Workshop on Challenges for Parallel Processing*, pages 8–15, August 1995.

[15] H. F. Salama, D. S. Reeves, and Y. Viniotis. The Delay-Constrained Minimum Spanning Tree problem. In *Proc. IEEE Symposium on Computers and Communications*, pages 699–703, July 1997.

[16] T. Tachikawa, H. Higaki, and M. Takizawa. Group communication protocol for realtime applications. In *Proc. 18th IEEE Intl. Conf. Distributed Computing Systems*, pages 40–47, May 1998.

[17] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, March 1998.

[18] R. Thakur and A. Choudhary. All-to-all communication on meshes with wormhole routing. In *Proc. 8th Intl. Parallel Processing Symposium*, pages 561–565, April 1994.

[19] K. Verstoep, K. Langendoen, and H. Bal. Efficient reliable multicast on Myrinet. In *Proc. Intl. Conf. Parallel Processing*, volume III, pages 156–165, August 1996.

[20] C.-L. Wang, P. B. Bhat, and V. K. Prasanna. High-performance computing for vision. *Proceedings of the IEEE*, 84(7):931–946, July 1996.

Efficient Collective Communication in Distributed Heterogeneous Systems *

Prashanth B. Bhat [1]
Department of EE-Systems, EEB 246
University of Southern California
Los Angeles, CA 90089-2562
prabhat@halcyon.usc.edu
Phone: (213)740-2350
Fax: (213)740-4418

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
raghu@aero.org
Phone: (310)336-1686
Fax: (310)336-4402

and

Viktor K. Prasanna [1]
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
prasanna@halcyon.usc.edu
Phone: (213)740-4483
Fax: (213)740-4418

*A preliminary version of this paper appears in the Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS 1999).

[1] Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

Abstract

With recent advances in high-speed networks, distributed heterogeneous computing has emerged as an attractive computational paradigm. Wide-area grid infrastructures will enable distributed applications – such as video conferencing and distributed interactive simulation – to seamlessly integrate collections of heterogeneous workstations, multiprocessors, and mobile nodes. The underlying network is typically a collection of several heterogeneous links, of different networking technologies. Such a heterogeneous network is also typical in local area workstation clusters, which are increasingly being used as alternatives to parallel computing systems. This paper introduces a framework for developing efficient collective communication schedules over such heterogeneous networks. We focus on application-level communication, between processes of a parallel program. Our framework consists of analytical models of the heterogeneous system, scheduling algorithms for the collective communication pattern, and performance evaluation mechanisms. We show that previous models, which considered node heterogeneity but ignored network heterogeneity, can lead to solutions which are worse than the optimal by an unbounded factor. We then introduce an enhanced communication model, and develop three heuristic algorithms for the broadcast and multicast patterns. The completion time of the schedule is chosen as the performance metric. The heuristic algorithms are FEF (Fastest Edge First), ECEF (Earliest Completing Edge First), and ECEF with look-ahead. For small system sizes, we find the optimal solution using exhaustive search. Our simulation experiments indicate that the performance of our heuristic algorithms is close to optimal. For performance evaluation of larger systems, we have also developed a simple lower bound on the completion time. Our heuristic algorithms achieve significant performance improvements over previous approaches.

Keywords: Metacomputing, heterogeneous networks, collective communication scheduling, broadcast, multicast.

Running Head: Communication Schedules for Heterogeneous Systems

Contact Author:

Viktor K. Prasanna

Department of EE-Systems, EEB 200C

University of Southern California

Los Angeles, CA 90089-2562

Email: prasanna@halcyon.usc.edu

Phone: (213)740-4483, Fax: (213)740-4418

1 Introduction

With recent advances in high-speed networks, distributed heterogeneous computing has emerged as an attractive computational paradigm. Computational grids [4] are emerging as an infrastructure that will connect distributed computational sites worldwide. This infrastructure would create a universal source of computing power, thereby providing pervasive and inexpensive access to advanced computational capabilities. A typical grid-based distributed computing system will consist of a collection of heterogeneous workstations, multiprocessors, and mobile nodes. These nodes communicate with one another using a common set of protocols over different types of communication links, such as ATM, FDDI, Ethernet, and wireless channels. An example of such a system is shown in Figure 1. Such a distributed computing system is heterogeneous both in the computing nodes and in the communication network.

The availability of high-speed wide-area networks has also enabled collaborative multimedia applications such as video conferencing, distributed interactive simulation, and collaborative visualization. For example, the *FACE* project [20] organized world-wide teleconferences among agents in Japan, USA, and the UK. The participating sites in these applications exchange large volumes of multimedia data, such as voice and video. Using the Internet, messages were propagated in about 60 *msec* between sites in Japan, while it took about 240 *msec* between Japan and Europe [20].

is parallelized across a local area workstation cluster. Several

Cluster computing is yet another paradigm that has been motivated by the availability of fast networks. The performance of recent networking technologies such as Fibre-Channel, Gigabit Ethernet, and ATM is comparable to that of the interconnection networks within high-end parallel computers. Applications can therefore be parallelized over a local area cluster, at a cost much lesser than that of a parallel machine. Due to the variety of networking technologies available, typical local area networks consist of several kinds of links. Communication among application processes would therefore be performed over this heterogeneous collection of networks. Recent research efforts have investigated techniques to improve communication performance in such a scenario. Two techniques were investigated by researchers at the University of Minnesota. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks [11, 12]. Experimental evaluation of different networking technologies in a NOW environment was also done at the Ohio State University. Networking technologies such as SCRAMNet, Fast Ethernet, and Myrinet were individually evaluated by sending a message between two nodes. It was observed that no single networking technology is the best for all message lengths. Researchers are investigating whether multiple networking technologies with complimentary features can be used together to improve the overall performance [10, 15].

In each of the above scenarios, *viz.,* distributed high performance computing, collaborative multimedia applications, and heterogeneous workstation clusters, it is extremely important to efficiently perform group communication over a heterogeneous network. Typical group communication patterns are multicast, broadcast, and total exchange. In the multicast pattern, a source node sends the same message to a subset of nodes in the system. The broadcast pattern is a special case of multicast where the message is sent from a source to all the other nodes. In the total exchange communication pattern, every node sends a distinct message to every other node. The goal is to optimize a specified performance measure, *eg.,* minimize the time at which all the messages have been delivered.

Several research projects, such as MSHN [16], Globus [8], and Legion [9], are developing toolkits and infrastructure support to enable the use of distributed heterogeneous systems for high performance computing. The issue of data dissemination middleware for wide-area network collaboratories is also being investigated [14]. Our research is a part of the MSHN project [16], which is a collaborative effort between DoD (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). MSHN (Management System for Heterogeneous Networks) is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. The goal is to schedule shared compute and network resources among individual applications so that their QoS requirements are satisfied.

In this paper, we develop efficient algorithms for broadcast and multicast in heterogeneous computing environments. These communication patterns occur in several scientific, commercial, as well as military applications. In the battlefield, rapid dissemination of work orders, threat scenarios, and emergency messages is critical [21]. A global satellite and ground-based networks are used in military battlefield to broadcast messages. The satellite sends the message to a group of base stations as it passes over them. The base stations then co-operatively broadcast the message to the other destinations over ground-based networks. In the past, broadcast and multicast problems have been studied extensively in the context of homogeneous and worm-hole routed networks [17]. Similarly, multicast protocols such as CBT, DVMRP, and PIM are now being deployed in wide-area networks. However, these techniques are not appropriate for the distributed network scenarios that we consider in this paper. For example, flooding is a technique where a node simultaneously sends the broadcast message to all its neighbors. The receiving nodes "flood" their neighbors in turn, until the message is received by all nodes. Some of the nodes could receive the message multiple times, depending on the network topology. Such techniques will not be efficient in wide-area heterogeneous networks, since each point-to-point communication event incurs an additional communication cost. Further, this will also introduce extra network congestion.

Recent research efforts [2] have investigated the problem of efficient broadcast and multicast in a network of heterogeneous workstations. The heterogeneity in the communication capabilities of the workstations was represented by associating a message initiation cost with each workstation. However, heterogeneity in the network was not considered. Based on this communication cost model, heuristic algorithms were developed for the broadcast and multicast problems. The heuristics achieve near-optimal performance for up to 10 nodes. In Section 2, we show that such a communication model can be very ineffective in a system with a heterogeneous network. We give examples where the completion time of a broadcast schedule using such a model is larger than the optimal completion time by an unbounded factor. It is therefore necessary to use a communication framework that considers heterogeneity in both the nodes and network links. Section 3 introduces our new communication model and framework. Our model represents the communication cost between two nodes P_i and P_j using two parameters: (i) a start-up time which accounts for the message initiation cost at P_i, and the network latency from P_i to P_j, and (ii) a data transmission cost which depends on the message size and the bandwidth from P_i to P_j. Using this model, we can consider the distributed system to be a fully connected network with a communication cost $\mathbf{C}_{ij}$ between every pair of nodes P_i and P_j. We do not assume a symmetric network, *i.e.,* $\mathbf{C}_{ij} \neq \mathbf{C}_{ji}$.

Since the problem of finding the optimal broadcast schedule in such a heterogeneous system is NP-complete, we have developed heuristic algorithms based on our communication framework. Our heuristic algorithms produce near optimal solutions for up to 10 nodes when tested with random networks. For larger size systems, it is extremely time consuming to compute the optimal solution. We have therefore developed a lower bound on the completion time. We evaluate the different heuristics by comparing their completion time with the lower bound.

The rest of the paper is organized as follows. In Section 2, we discuss related work and its shortcomings. Section 3 presents our formal model and general framework for collective communication in heterogeneous distributed computing environments. In Section 4, we present several heuristic algorithms for the broadcast and multicast problems. Section 6 compares the performance of our heuristics with previous algorithms, using simulation results. Section 7 identifies future research directions.

2 Shortcomings of Previous Research

Collective communication in homogeneous workstation networks and tightly coupled parallel systems has been thoroughly researched over the years. Communication libraries for frequently used patterns such as *total exchange, one-to-all broadcast, all-to-all broadcast,* and *gather* have been developed [1, 3, 22, 23].

However, collective communication in heterogeneous systems has not been investigated until very recently [13, 2]. The Efficient Collective Operations (ECO) [13] package was developed for networks of heterogeneous workstations. It implements the same functionality as the collective communication suite in the MPI standard. The ECO approach consists of first partitioning the network into subnets. A subnet consists of hosts which are in the same physical network. The collective communication then proceeds in two phases, inter-subnet and intra-subnet. However, such a two-phase strategy does not always ensure efficient implementations of collective communication patterns. This is especially true if the the inter-subnet links are much slower than the intra-subnet links. Implementations of MPI on the Grid have also been investigated [5]. However, the problem of efficiently implementing collective communication was not studied in detail.

Banikazemi *et. al.* [2] identified the important problem of performing efficient broadcast and multicast among a cluster of heterogeneous workstations. A homogeneous network was assumed. Their communication model associates a message initiation cost T_i with each of the P workstations. T_i is incurred whenever the i^{th} workstation (P_i) sends a message, independent of the identity of the receiving workstation. Based on this communication cost model, it was shown that broadcast schedules based on binomial trees, which achieve good results in homogeneous systems, can be very ineffective. A $P-1$ step heuristic algorithm, called Fastest Node First (FNF), was developed. Each step of the heuristic selects a sender and a receiver. The receiver is the node with the lowest T_i among the remaining receivers.

The sender is the node that can complete the communication event at the earliest possible time. The FNF heuristic was evaluated for systems with up to 10 nodes [2]. For the examples considered, the completion time of the FNF heuristic was very close to the optimal.

However, there are scenarios where the performance of the FNF heuristic can be sub-optimal. Consider the example where the source has cost 1, there are n nodes with costs $n, n+1, n+2, \ldots, 2n-1$, and $2n$ slow nodes with very high costs. In the optimal schedule, the source would first send n messages to nodes with cost $2n-1, 2n-2, \ldots,$ respectively. At time n, the node with cost n has received the message from the source. Immediately after receiving the message, each of these nodes initiates a message to one of the slow nodes. During the time interval $[n, 2n]$, the source sends n more messages to the remaining slow nodes. The schedule completes at time $2n$.

In the FNF schedule, the source will send messages to nodes with cost $n, n+1, \ldots, 2n-1$ respectively. At time n, n nodes will have received the message. If each node immediately initiates a new message, each of the nodes with costs n to $\frac{3n}{2}$ can reach a slow node by time $2n$. During the time interval $[n, 2n]$, the source sends n more messages to n of the slow nodes. Thus, at time $2n$, $\frac{n}{2}$ of the slow nodes have not yet received the message. The schedule takes $\frac{n}{8}$ extra time units to complete. For large values of n, the completion time of the FNF schedule is much larger than the optimal.

A more significant shortcoming of [2] was the assumption of the homogeneous network. In a typical heterogeneous system, the communication cost depends both on the communication capability of the workstations as well as the network performance. Our paper investigates the impact of heterogeneity in both these aspects. We first illustrate the importance of considering network heterogeneity, using an example. Consider a system with 3 nodes, and pairwise communication costs as shown in Eq (1). The $(i,j)^{th}$ entry of $\mathbf{C}$ $(0 \leq i,j < 3)$ denotes the time to send the broadcast message from node P_i to P_j. This includes the message initiation cost on node P_i and also the network latency from P_i to P_j. Section 3 discusses this communication model in detail. Node P_0 is the source.

$$\mathbf{C} = \begin{bmatrix} 0 & 10 & 995 \\ 2000 & 0 & 10 \\ 70 & 5 & 0 \end{bmatrix} \tag{1}$$

To develop a communication schedule based on node heterogeneity alone, we associate a communication cost T_i with each node. This is calculated as the average send cost from node P_i to all the other nodes. Thus, in Eq (1), $T_0 = 335, T_1 = 670, T_2 = 25$. We can now use the FNF heuristic [2] for this problem. Since the heuristic operates on a modified version of the input data (*i.e.,* the average communication costs), we call this the modified FNF heuristic.

For the example of Eq (1), the heuristic begins with node P_0 as the only sender. In the first step, P_2 is selected as the receiver. The communication from P_0 to P_2 takes 995 time units. Both these nodes are ready to send the next message at time 995. In the next step, node P_2 is selected as the sender and P_1 is selected as the receiver. This communication event takes 5 time units. The broadcast therefore takes 1000 time units to complete. Figure 2(a) shows this communication schedule.

However, it is easy to see that the optimal schedule takes only 20 time units. In the first step, P_0 sends a message to P_1 in 10 time units. In the next step, P_1 sends a message to P_2 in 10 time units. This schedule is shown in Figure 2(b). Thus, the use of a single message initiation cost for each node results in a communication schedule which is 50 times worse than the optimal schedule for this example. In the above example, we used the average send cost from each sender as its communication cost. Alternatively, we could have used the minimum send cost of each sender as its communication cost T_i. In Eq (1), the costs would then be $T_0 = 10, T_1 = 10, T_2 = 5$. It can be easily verified that the modified FNF heuristic again takes 1000 time units to complete.

The performance of the modified FNF heuristic would be still worse if the value of $C_{2,0}$ was larger. For example, if $C_{2,0}$ was 9995 instead of 995, the completion time would have been 10000 time units, *i.e.* 500 times the optimal completion time. We summarize this observation in the following lemma.

Lemma 1 *In the presence of a heterogeneous network, there exist input instances for which the ratio of the completion time of the modified FNF heuristic to the optimal completion time is unbounded.*

Thus, communication models which consider only node heterogeneity can result in arbitrarily bad performance. It is therefore important to consider both node heterogeneity and network heterogeneity when designing communication algorithms for the broadcast problem.

3 A Communication Framework for Distributed Heterogeneous Systems

We now present our communication scheduling framework for distributed heterogeneous systems. The framework consists of four main components: (a) A communication model, (b) A scheduling formalism, (c) Scheduling heuristics, and (d) Performance metrics. In this section, we describe an enhanced communication model which incorporates node and network heterogeneity. Section 4 describes our formalism and heuristic algorithms for broadcast and multicast based on this model. The performance metric used in this paper is the completion time. Other candidate metrics are discussed in Section 8.

3.1 Communication Model

Consider a distributed heterogeneous system (Figure 1) with P nodes. We represent the computing nodes and network links in such a system using a directed graph G with P vertices. An edge (v_i, v_j) in G represents the path between nodes P_i and P_j, which could include links from multiple networks of different latencies and bandwidths. The weight $\mathbf{C}_{ij}$ of edge $(v_i, v_j), (0 \leq i, j < P)$ represents the time to send the broadcast message from P_i to P_j. If there exists at least one path between every pair of nodes in the system, G will be a complete graph. The graph is not necessarily symmetric, *i.e.* $\mathbf{C}_{ij} \neq \mathbf{C}_{ji}$, in general. Figure 3 shows an example graph G with $P = 5$ nodes. The graph is symmetric in this example. The information can also be represented as a $P \times P$ communication matrix $\mathbf{C}$, with entries $\mathbf{C}_{ij}$, as shown in Eq (1).

Our communication model represents the network performance between any processor pair (P_i, P_j) using two parameters: a start-up cost T_{ij} and a data transmission rate B_{ij}. The time for sending a m byte message between these nodes is given by $T_{ij} + \frac{m}{B_{ij}}$. A similar communication model has been widely used for tightly-coupled homogeneous distributed memory systems with good results [24]. In networked heterogeneous systems, typical values for the start-up cost could be in the range of 10 to 500 μs, while typical values for the bandwidth could be in the range of kb/s to hundreds of Mb/s. Note that the communication time depends on the identities of both the sender and receiver, unlike previous models [2]. The model thus enables a realistic estimate of the communication time between any pair of nodes.

Tables 1 and 2 are examples of measured network performance on the GUSTO testbed of the Globus distributed heterogeneous system [6]. The table shows five of the GUSTO sites: NASA AMES, Argonne National Lab, University of Indiana, USC-ISI, and NCSA. Observe that the network performance varies considerably between different pairs of nodes, and depends on both the source and destination. For instance, the bandwidth between USC-ISI and AMES is much larger than the bandwidth between USC-ISI and IND. Previous communication models [2], which assume that the communication time from node P_i to node P_j is independent of P_j and depends only on the source node P_i, are therefore unlikely to be effective for such systems.

Our model assumes that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple messages to be transmitted simultaneously. Software support for non-blocking and multithreaded communication sometimes allows applications to initiate multiple send and receive operations. However, all these operations are eventually serialized by the single hardware port to the network. Our model accurately represents this phenomenon.

If multiple nodes simultaneously send to any node P_j, we say that node contention occurs at P_j. The model assumes that these messages are received one after the other at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation.

Based on the network performance parameters and our communication model, we can calculate the communication time to send the broadcast message between any pair of nodes in the heterogeneous network. This information is used to determine the edge weights of G and the entries of the communication matrix **C**. The communication matrix for broadcasting a 10 MByte message over the network of Tables 1 and 2 is shown in Eq (2). Entries are in *sec.*

$$\mathbf{C} = \begin{bmatrix} 0 & 156 & 325 & 39 & 205 \\ 156 & 0 & 163 & 115 & 33 \\ 325 & 163 & 0 & 257 & 179 \\ 39 & 115 & 257 & 0 & 16 \\ 205 & 33 & 179 & 16 & 0 \end{bmatrix} \qquad (2)$$

4 Heuristics for Broadcast and Multicast

Our algorithms for the broadcast and multicast problems can be described using a set-based formalism. The P nodes are partitioned into three sets, **A**, **B**, and **I**. At any time, set **A** consists of nodes which have already received the message. Set **B** consists of nodes which must receive the message in the future. **I** contains the other nodes. Initially, set **A** consists of the source node while set **B** consists of the destination nodes for the multicast, *i.e.* **B**=**D**. For the broadcast problem, $\mathbf{I} = \phi$.

At every step, a sender from **A** and a receiver from **B** are chosen. For the multicast problem, the message could also be relayed through one of the nodes in **I**, if this path incurs lower communication time. After each communication event, the receiver node (and the intermediate node, if one was chosen) is moved to **A**. The communication schedule involves $\mid \mathbf{D} \mid$ such steps. We now present the baseline algorithm and our FEF, ECEF, and look-ahead heuristics.

Baseline Algorithm:

We use the modified FNF heuristic [2] as a baseline algorithm. This algorithm associates a single communication cost with each node rather than a distinct cost for each pair of nodes. We use the average send cost from node P_i to all the other nodes as its communication cost T_i. The FNF heuristic algorithm [2] consists of $P-1$ steps. At every communication step, the node from **B** with the lowest T_j is chosen as the receiver. A sender is chosen such that the communication event can be completed at the earliest possible time. This is the node P_i which has the minimum value of

$$R_i + T_i \tag{3}$$

where R_i is the ready time of the sender P_i.

Fastest Edge First (FEF):.

Each step of our FEF heuristic selects the smallest weight edge (i, j) where P_i belongs to **A** and P_j belongs to **B**. The choice of the edge determines both the sender and receiver node for the corresponding communication step. P_j is then moved from **B** to **A**. The communication step starts at R_i, and takes $\mathbf{C}_{i,j}$ time units. During this time, both P_i and P_j are busy.

The algorithm initially sorts the outgoing edges from each node in increasing order of their weights. This phase takes $O(P^2 \log P)$ time. The senders in **A** are then sorted in increasing order of their minimum weight outgoing edge. The new node added to set **A** at every step is inserted into the sorted sender list based on its minimum weight outgoing edge. The algorithm terminates after all the destination nodes have been moved to **A**. This involves $P - 1$ steps for the broadcast algorithm and a maximum of $P - 1$ steps for the multicast algorithm. The running time for this phase is also $O(P^2 \log P)$. The overall running time of the FEF heuristic is therefore $O(P^2 \log P)$.

Figure 4 shows the steps in the FEF heuristic for the broadcast problem in the five node system of Figure 3. Figure 4(a) shows the initial situation when set **A** contains only the source node, and set **B** contains the other nodes. The figures show the edge weights of only the edges in the **A-B** cut. Figures 4(b)-4(d) show the sequence in which the FEF heuristic moves edges from **B** to **A**. Figure 5 shows the broadcast tree for this schedule.

Earliest Completing Edge First (ECEF):

The structure of our ECEF heuristic algorithm is similar to the FEF heuristic. At every step, an edge (i, j) is selected, where P_i belongs to **A** and P_j belongs to **B**. The choice of the edge considers both the weight of the edge and the ready time of the sender. The chosen communication event is the one that can complete earliest. Thus, the chosen edge is the one that minimizes the sum

$$R_i + \mathbf{C}_{i,j} \tag{4}$$

over all senders P_i and receivers P_j, where R_i is the ready time of sender P_i. As in the

FEF heuristic, a sorted list of senders is maintained. The senders are sorted based on both their ready time and their minimum weight outgoing edge. The heuristic has a running time of $O(P^2 \log P)$.

Look-ahead Algorithm:

Our look-ahead algorithm is an enhanced version of the ECEF heuristic. At each step of the heuristic, a *look-ahead value* L_j is calculated for each node P_j in $\mathbf{B}$. This value quantifies the "goodness" of moving node P_j from $\mathbf{B}$ to $\mathbf{A}$. At each step, the algorithm first computes the value of L_j for all nodes in $\mathbf{B}$. As in the ECEF heuristic, an edge is then selected from the $\mathbf{A}$-$\mathbf{B}$ cut. The chosen edge is the one that minimizes the sum

$$R_i + \mathbf{C}_{i,j} + L_j \tag{5}$$

The look-ahead function can be defined in several ways. We have used the following look-ahead measure.

$$L_j = \min_{P_k \in \mathbf{B}} \mathbf{C}_{j,k} \tag{6}$$

Thus, for a given node P_j in $\mathbf{B}$, the minimum communication cost from itself to all the other nodes in $\mathbf{B}$ is used as the look-ahead value. Intuitively, such a look-ahead function increases the usefulness of P_j as a sender, if it is moved to $\mathbf{A}$.

The running time of the look-ahead algorithm is $O(P^3)$, since the evaluation of the look-ahead measure for each element of $\mathbf{B}$ at every step takes $O(P)$. Alternative look-ahead functions can also be used, such as the average of the communication costs from P_j to other nodes in $\mathbf{B}$. L_j could also be calculated as the average cost of senders to receivers, assuming that P_j is made a sender. This look-ahead function has a computational complexity of $O(P^2)$, and the overall running time will therefore be $O(P^4)$. Our experiments in Section 6 use the look-ahead measure of Eq (6).

5 Performance Evaluation

We evaluate the performance of our heuristic algorithms using a simulator. For performance comparison, the modified FNF algorithm is used as a baseline algorithm. We also define below a lower bound on any communication schedule for the broadcast and multicast problems. For small system sizes, the optimal schedule is also computed.

5.1 A Lower Bound

Let P_0 be the source of the broadcast or multicast operation, and $\mathbf{D}$ represent the set of destination nodes. $\mathbf{D} \subset \{P_1, P_2, \ldots, P_{P-1}\}$ for multicast, while $\mathbf{D} = \{P_1, P_2, \ldots, P_{P-1}\}$ for broadcast. For each node P_i in $\mathbf{D}$, we can compute the shortest path from the source node P_0 to P_i. The weight of this path represents the earliest time at which the broadcast message from P_0 can reach P_i. This is therefore called the *Earliest Reach Time* of node P_i, denoted as ERT_i.

Lemma 2 *A lower bound on any communication schedule for the broadcast or multicast problem is given by*

$$LB = \max_{P_i \in \mathbf{D}} ERT_i \tag{7}$$

Proof: We know that ERT_i represents the earliest time at which node P_i can be reached. From the definition of the broadcast and multicast communication pattern, the message must reach every node in **D**. Hence, no communication schedule can complete until the node with the maximum ERT is reached. Eq (7) therefore gives a lower bound on the completion time. □

The lower bound is not tight, since it assumes that the messages from the source to each destination can proceed in parallel. Thus, the optimal completion time could be significantly larger than the lower bound.

Lemma 3 *For any instance of the multicast or broadcast problem, the optimal completion time is bounded by* $\mid \mathbf{D} \mid \times LB$*,* i.e.,

$$\frac{OptimalCompletionTime}{LB} \leq \mid \mathbf{D} \mid \tag{8}$$

Further, this ratio is tight.

Proof: The lower bound LB of Eq (7) is the communication time to send the message from the source to the farthest node. Thus, the communication time to send a message from the source to any node is $\leq LB$. We can always construct a communication schedule in which the source sequentially sends $\mid \mathbf{D} \mid$ messages to all the destinations. The $\mid \mathbf{D} \mid$ communication steps can therefore be completed in atmost $LB \times \mid \mathbf{D} \mid$ time units.

To prove that the ratio is tight, consider the broadcast problem on the communication cost matrix of Eq (9).

$$\mathbf{C} = \begin{bmatrix} 0 & 10 & 10 & \ldots & 10 \\ 100 & 0 & 100 & \ldots & 100 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 100 & 100 & 100 & \ldots & 0 \end{bmatrix} \tag{9}$$

In this matrix, $\mathbf{C}_{0j} = 10, (0 < j < P)$. Also, $\mathbf{C}_{ij} = 100, (0 < i < P, i \neq j)$. The diagonal entries $\mathbf{C}_{ii} = 0, (0 \leq i < P)$. The shortest path to every node P_i is the direct path (P_0, P_i). The lower bound would be the maximum outgoing edge from P_0, *i.e.,* 10. However, the optimal schedule has a completion time of $10 \mid \mathbf{D} \mid$. Thus, there exist examples wherein the optimal completion time is $\mid \mathbf{D} \mid$ times as large as our simple lower bound. □

5.2 Computing the Optimal Schedule

The possible number of communication schedules for a broadcast or multicast problem instance with P nodes is exponential in P. The completion times of these schedules can vary considerably, depending on the performance of the heterogeneous network links. Finding the optimal communication schedule is an NP-Complete problem. However, for systems with a small number of nodes, we can find the optimal schedule using exhaustive search. Our algorithm, which uses a branch-and-bound strategy, computes the optimal solution for up to 10 nodes in a reasonable amount of time. For small system sizes, we shall compare the performance of our heuristic algorithms with the optimal solution.

6 Experimental Results

We have developed a software simulator that executes the heuristic algorithms of Section 4, and calculates the completion time for each of them. The inputs to the simulator are the number of nodes, the size of the message to be broadcast or multicast, and the range of start-up times and bandwidths in the heterogeneous network. The simulator generates a random communication matrix based on these parameters. For the case of multicast, the number of destinations is given as input, and the simulator randomly chooses destination nodes. The simulator then executes the steps in the heuristic algorithms for 1000 random input configurations. Finally, the simulator reports the average completion time for each heuristic.

Figures 6 and 7 compare the performance of the different communication scheduling heuristics for the broadcast problem with a message size of 1 MB. The pairwise network latencies and bandwidths are chosen in the ranges of 10 μsec to 1 $msec$, and $10kB/s$ to $200MB/s$ respectively. The graph shows the completion time for the baseline algorithm, the FEF, ECEF, and look-ahead heuristics, and our simple lower bound. For small system sizes (upto 10 nodes), the optimal completion time is also shown. Since our lower bound is not tight, it is typically much lesser than the optimal completion time. The graph shows that the completion time of our heuristic algorithms is always close to the optimal. The ECEF and look-ahead algorithms have a lower completion time than that of the FEF heuristic. The completion time of the baseline algorithm is significantly larger than that of the other heuristics. This shows the benefit of using a communication model which accurately represents heterogeneity in the network, as well as in the nodes.

The performance advantage of our heuristic algorithms over the baseline algorithm can also be seen in Figure 8. Figures 8 and 9 consider a system with two distinct geographically distributed clusters. It is assumed that half the nodes are in the first cluster, while the other nodes are in the second cluster. The heterogeneous network is assumed to be fast within each cluster, but is slow across clusters. For the intra-cluster networks, the latencies and bandwidths are in the ranges of 10 μsec to 1 $msec$, and $10MB/s$ to $200MB/s$ respectively. For the inter-cluster networks, the latencies and bandwidths are in the ranges of $1msec$ to

$20msec$, and $10kB/s$ to $50kB/s$ respectively. As before, the size of the broadcast message is 1 MB.

Figure 10 shows the completion time for multicast in a 100 node system. The number of multicast destinations is increased from 5 to 90. For the case of k destinations, 1000 experiments are performed with k randomly chosen destinations. The average completion time is plotted in Figure 10. Observe that the heuristic algorithms again significantly outperform the baseline algorithm.

7 Research Issues

The experimental results of Section 6 clearly show the performance benefits of our heuristic algorithms. However, there are scenarios in which some of our heuristics can have poor performance. Consider the asymmetric communication cost matrix of Eq (10), which could be a system with Asymmetric Digital Subscriber Lines (ADSL).

$$\mathbf{C} = \begin{bmatrix} 0 & 2 & 2 & 2.1 & 2 \\ 100 & 0 & 100 & 100 & 100 \\ 100 & 100 & 0 & 100 & 100 \\ 1000 & 0.1 & 0.1 & 0 & 0.1 \\ 100 & 100 & 100 & 100 & 0 \end{bmatrix} \tag{10}$$

In the optimal broadcast schedule, P_0 sends the message to P_3 in step 1, and then P_3 then sends messages to the other nodes in steps 2, 3, and 4. This has a completion time of 2.4 time units. However, the ECEF heuristic sends the message from P_0 to P_1 in step 1,P_0 to P_2 in step 2, P_0 to P_4 in step 3, and P_0 to P_3 in step 4. The completion time is 8.4 time units. The look-ahead algorithm does find the optimal schedule. It chooses the node P_3 as the receiver in the first step, since P_3 has a low-cost outgoing edge.

However, the performance of the look-ahead schedule is poor for the communication matrix of Eq (11). The algorithm takes 4.1 time units (P_0 to P_1, P_1 to P_2, P_2 to P_3, and P_3 to P_4). The optimal schedule takes only $2.2 + 2\epsilon$ time units (P_0 to P_3, P_3 to P_4, P_4 to P_1, P_4 to P_2). For larger systems, the difference between the completion times of the look-ahead and optimal schedules can be much higher.

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 1 & 1.1 & 100 \\ 100 & 0 & 1 & 100 & 100 \\ 100 & 100 & 0 & 1 & 100 \\ 100 & 100 & 100 & 0 & 1.1 \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 \end{bmatrix} \tag{11}$$

However, communication matrices such as Eq (11) do not typically occur in real scenarios. Often, $\mathbf{C}$ is symmetric. The triangle inequality is also usually valid, *i.e.*,

$$\mathbf{C}_{ij} \leq \mathbf{C}_{ik} + \mathbf{C}_{kj}, 0 \leq k < P \tag{12}$$

For such a system, stronger performance bounds than Eq (8) could be shown. Proving these performance bounds is an interesting research direction.

It is also interesting to explore new heuristic schedules based on the Minimum Spanning Tree(MST) and Steiner Tree algorithms. The steps in our FEF algorithm are identical to Prim's MST algorithm [18]. We are currently investigating a *progressive MST* approach. This is an enhancement to Prim's algorithm which accounts for the ready time of each node. After each step of the algorithm, some of the edge weights are updated to reflect the change in ready times. We are also investigating a *two-phase* approach. During the first phase, a MST is constructed. The structure of the MST is used to guide the selection of intermediate nodes for the second phase, which constructs the heuristic schedule.

The main difference between the MST problem and our broadcast problem is the cost metric. The metric in the MST problems is usually the total weight of edges in the spanning tree. In contrast, the completion time of the broadcast and multicast problems is the time at which all nodes have received the message. Delay-constrained MST problems, which minimize the maximum delay between the source and any destination, have also been considered [19]. However, this metric is also different from the completion time. Consider the example of Eq (10). The delay-constrained algorithm would create a MST with edges $(P_0, P_1), (P_0, P_2), (P_0, P_3)$, and (P_0, P_4). Although the maximum delay is 2.1, the completion time is 8.1 time units. In fact, if the triangle inequality of Eq (12) holds, the delay-constrained algorithm will always send $\mid \mathbf{D} \mid$ messages sequentially from the source to each destination.

A second difference is that the widely known MST algorithms of Prim and Kruskal were developed for undirected graphs. Our progressive and two-phase techniques can build upon these techniques if the heterogeneous network is symmetric. For asymmetric networks, MST algorithms for directed graphs can be used [7].

In designing a heuristic, we must give special attention to two kinds of nodes: (a) Nodes which are hard to reach from every other node, and are also unable to reach other nodes quickly. The message to such a node should be sent early in the schedule, so that this communication event does not delay the completion time. (b) Nodes which are a little hard to reach, but which can reach many other nodes very easily. Such nodes should be selected early, so that they can relay the message to the other nodes.

An alternating *near-far* approach would balance these two conflicting goals. All nodes are initially sorted in increasing order of their ERT. In the first two steps, messages are sent to the nearest node (say P_i), and to the farthest node (say P_j). From this point onwards, P_i and its recipients will send messages to the *near* nodes. This group always selects the nearest unreached node at every step. P_j and its recipients will send messages to the *far* nodes. This group selects the farthest unreached node. Such a near-far strategy is likely to balance the two conflicting goals discussed above.

For the multicast problem, an enhanced algorithm which can relay messages through nodes in the intermediate set **I**, must be considered. Our current algorithm does not incorporate this aspect. The problem of scheduling multiple simultaneous multicasts will also be considered.

The previous sections have illustrated the use of our framework for a specific cost model and performance metric. We now discuss some variations and extensions of these components. Our communication model assumed that a node can send and receive atmost one message at any time. In a *non-blocking* communication model, this assumption is relaxed. After an initial start-up time, the sender can initiate a new message. The first message is completed by the network without further intervention by the sender. Thus, a node could send out several messages before the first message reaches the receiver. Similar assumptions can be made at the receiver too.

We have used the completion time as our performance metric. *Robustness* metrics can be used to measure the ability of a communication schedule to reach all destinations, inspite of intermediate node or link failures. A communication schedule could increase its robustness measure by sending redundant messages for fault tolerance. Alternatively, acknowledgement schemes and time-out parameters could be used to detect failures before resending a message over a different path. Another candidate metric is the *amount of transmitted data.*

8 Conclusion

Efficient communication support is extremely important for several distributed computing scenarios. Examples are collaborative multimedia applications, parallel computing over workstation clusters, and high performance computing over computational grids. This paper has introduced an analytical framework for designing efficient collective communication algorithms. The main components of our framework are a communication model to represent the heterogeneous network and nodes, a scheduling formalism, performance metrics, and scheduling algorithms. Based on this framework, we have developed efficient solutions for broadcast and multicast. We have also identified several promising research directions to extend our work. We believe that future work along these directions can accelerate the widespread use of distributed heterogeneous computing.

References

[1] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *IEEE Trans. Parallel and Distributed Systems*, 6(2):154–164, February 1995.

[2] Mohammad Banikazemi, Vijay Moorthy, and Dhabaleswar K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proc. Intl. Conf. Parallel Processing*, pages 460–467, 1998.

[3] J. Bruck, Danny Dolev, Ching-Tien Ho, Marcel-Catalin Rosu, and Ray Strong. Efficient Message Passing Interface (MPI) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, 40(1):19–34, January 1997.

[4] I. Foster and C. Kesselman, *Eds. The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1998.

[5] I. Foster and N. Karonis. A grid-enabled mpi: Message passing in heterogeneous distributed computing systems. In *Proc. SuperComputing Conference*, 1998.

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[7] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding Minimum Spanning Trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[8] Globus Web Page. *http://www.globus.org.*

[9] A. S. Grimshaw and W. A. Wulf. Legion - a view from 50,000 feet. In *Proc. Fifth IEEE Intl. Symp. on High Performance Distributed Computing*, August 1996.

[10] M. Jacunski, V. Moorthy, P. Ware, M. Pillai, D. K. Panda, and P. Sadayappan. Low latency message passing for reflective memory networks. In *Proc. Communication and Architectural Support for Network-Based Parallel Computing (CANPC)*, January 1999.

[11] JunSeong Kim and David J. Lilja. Exploiting multiple heterogeneous networks to reduce communication costs in parallel programs. In *Proc. Heterogeneous Computing Workshop*, pages 83–95, April 1997.

[12] JunSeong Kim and David J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intl. Symp. High Performance Distributed Computing*, 1997.

[13] Bruce B. Lowekamp and Adam Beguelin. ECO: Efficient Collective Operations for communication on heterogeneous networks. In *Proc. 10th Intl. Parallel Processing Symposium*, pages 399–405, April 1996.

[14] G. Robert Malan, Farnam Jahanian, and Peter Knoop. Comparison of two middleware data dissemination services in a wide-area distributed system. In *Proc. Intl. Conf. Distributed Computing Systems*, May 1997.

[15] V. Moorthy et al. Low latency message passing on workstation clusters using SCRAMNet. In *Proc. Intnl. Parallel Processing Symposium (IPPS)*, April 1999.

[16] MSHN Web Page. *http://www.mshn.org.*

[17] D. K. Panda. Issues in designing efficient and practical algorithms for collective communication on wormhole-routed systems. In *ICPP Workshop on Challenges for Parallel Processing*, pages 8–15, August 1995.

[18] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[19] Hussein F. Salama, Douglas S. Reeves, and Yannis Viniotis. The Delay-Constrained Minimum Spanning Tree problem. In *Proc. IEEE Symposium on Computers and Communications*, pages 699–703, July 1997.

[20] T. Tachikawa, H. Higaki, and M. Takizawa. Group communication protocol for realtime applications. In *Proc. 18th IEEE Intl. Conf. Distributed Computing Systems*, pages 40–47, May 1998.

[21] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, March 1998.

[22] Rajeev Thakur and Alok Choudhary. All-to-all communication on meshes with wormhole routing. In *Proc. 8th Intl. Parallel Processing Symposium*, pages 561–565, April 1994.

[23] K. Verstoep, K. Langendoen, and H. Bal. Efficient reliable multicast on Myrinet. In *Proc. Intl. Conf. Parallel Processing*, volume III, pages 156–165, August 1996.

[24] C.-L. Wang, P. B. Bhat, and V. K. Prasanna. High-performance computing for vision. *Proceedings of the IEEE*, 84(7):931–946, July 1996.

Author Biographies

Prashanth B. Bhat is a Ph.D. candidate in Computer Engineering at the University of Southern California, Los Angeles. He received his B.Tech. degree in Computer Engineering from the Karnataka Regional Engineering College, India, in 1992. He received his M.E. degree in Computer Science and Engineering from the Indian Institute of Science, Bangalore, in 1994. During the summer of 1998, he was a research intern at Hewlett-Packard laboratories, Palo Alto. His research interests include scheduling techniques for parallel and distributed systems, High Performance Computing and parallel computer architecture.

C. S. Raghavendra is with The Aerospace Corporation in El Segundo, California. From September 1982 to December 1991 he was on the faculty of Electrical Engineering-Systems Department at the University of Southern California, Los Angeles. From January 1992 to August 1997 he was on the faculty of the School of Electrical Engineering and Computer Science at the Washington State University in Pullman as the Boeing Centennial Chair Professor of Computer Engineering. He received the B.E and M.E degrees in Electronics and Communication from the Indian Institute of Science, Bangalore, in 1976 and 1978, respectively. He received the Ph.D degree in Computer Science from the University of California at Los Angeles in 1982. His research interests are wireless networks, parallel processing, fault tolerant computing, and distributed systems. Dr. Raghavendra is a recipient of the Presidential Young Investigator Award for 1985 and became a Fellow of the IEEE in 1997.

Viktor K. Prasanna (V.K. Prasanna Kumar) received his B.S. in Electronics Engineering from the Bangalore University and his M.S. from the School of Automation, Indian Institute of Science. He obtained his Ph.D. in Computer Science from Pennsylvania State University in 1983. Currently, he is a Professor in the Department of Electrical Engineering - Systems, University of Southern California, Los Angeles and serves as the Director of the Computer Engineering Division. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision.

Dr. Prasanna has published extensively and consulted for industries in the above areas. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He was the general chair of the IEEE International Parallel Processing Symposium, 1998 and has been the key person in developing the meeting into a premier international meeting in parallel computing. He also serves on the editorial boards of the Journal of Parallel and Distributed Computing and IEEE Transactions on Computers. He is the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a Fellow of the IEEE.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		34.5	89.5	12	42
ANL	34.5		20	26.5	4.5
IND	89.5	20		42.5	21.5
USC-ISI	12	26.5	42.5		29.5
NCSA	42	4.5	21.5	29.5	

Table 1: Latency (ms) between 5 GUSTO sites.

	AMES	ANL	IND	USC-ISI	NCSA
AMES		512	246	2044	391
ANL	512		491	693	2402
IND	246	491		311	448
USC-ISI	2044	693	311		4976
NCSA	391	2402	448	4976	

Table 2: Bandwidth (kbits/s) between 5 GUSTO sites.

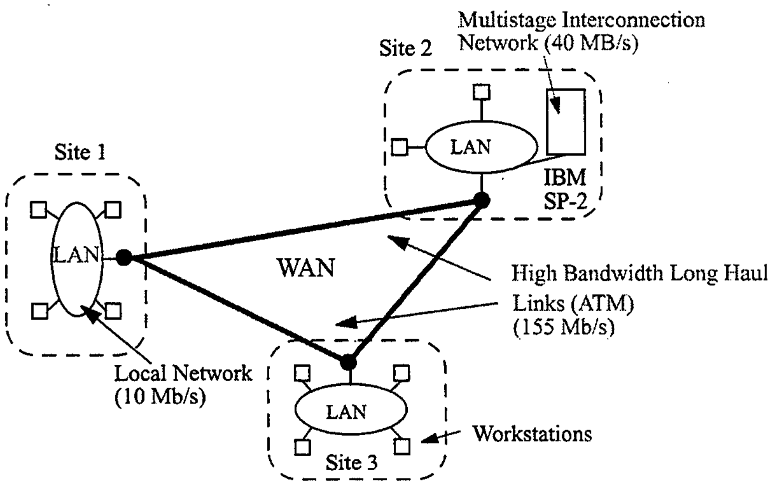

Multistage Interconnection Network (40 MB/s)
Site 2
LAN
IBM SP-2
Site 1
LAN
WAN
High Bandwidth Long Haul Links (ATM) (155 Mb/s)
Local Network (10 Mb/s)
LAN
Workstations
Site 3

Figure 1: A typical distributed heterogeneous system.

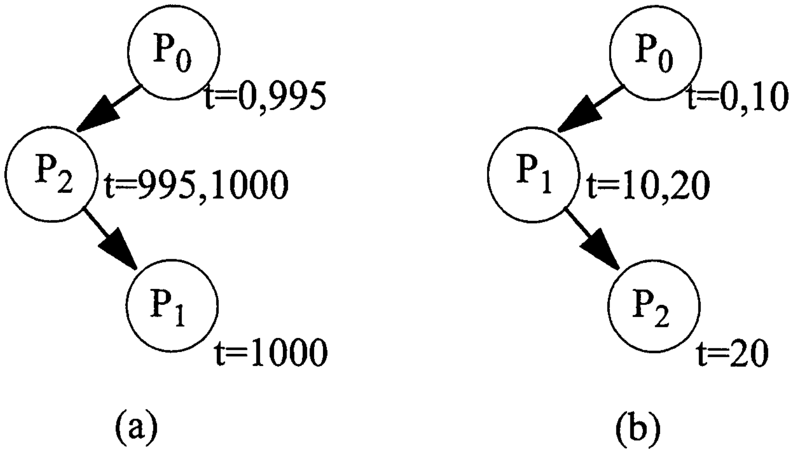


Figure 2: Broadcast schedules for the example in Eq (1): (a) Modified FNF schedule (b) Optimal schedule. The time values at each node represent the times at which the node is ready to send to another destination.

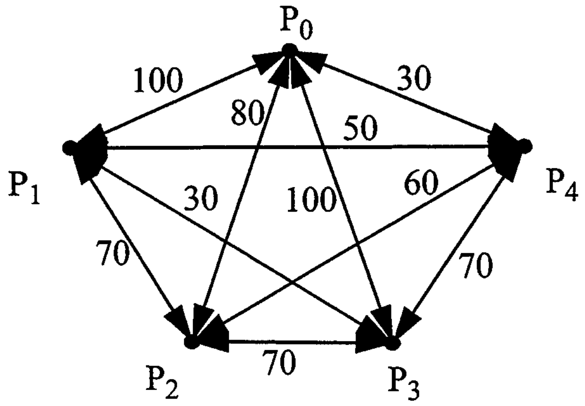


Figure 3: Directed graph G for a 5 node system.

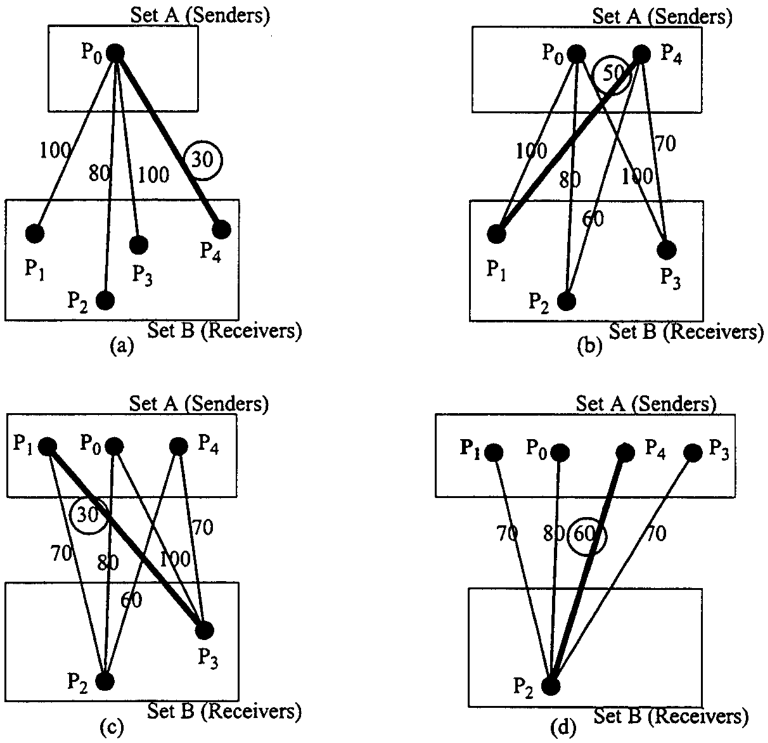


Figure 4: Steps in the FEF communication schedule for the 5 node heterogeneous system of Figure 3.

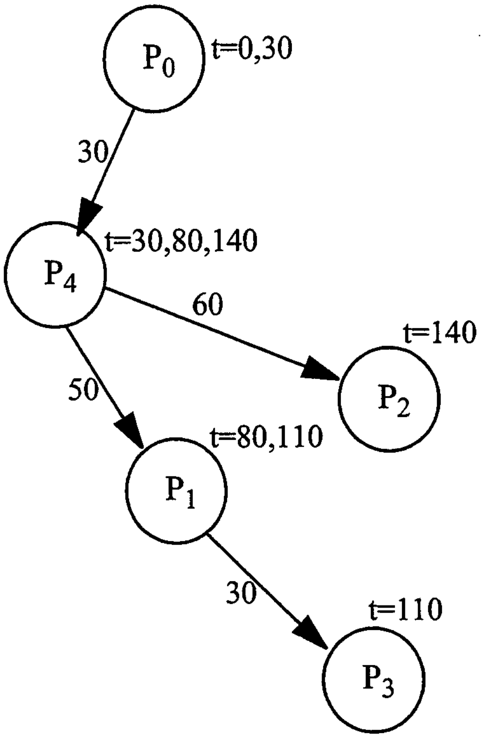


Figure 5: Broadcast tree for the FEF communication schedule of Figure 4. The time values at each node represent the times at which the node is ready to send to another destination.

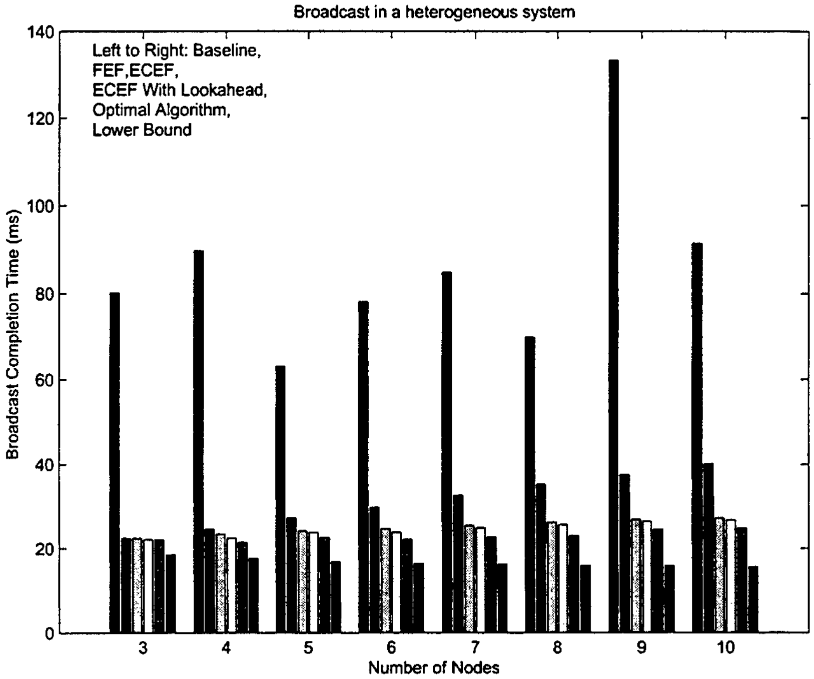


Figure 6: Simulation results for broadcast in a heterogeneous system.

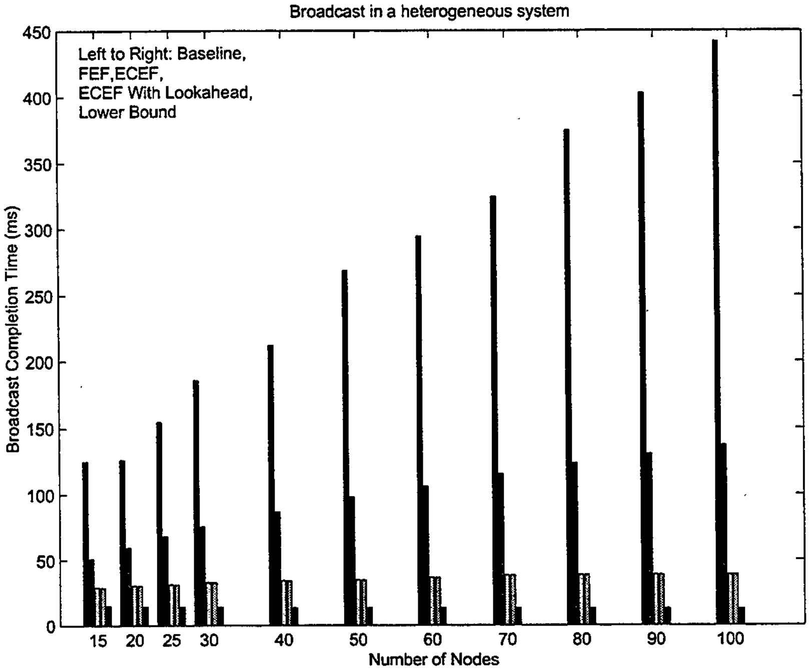


Figure 7: Simulation results for broadcast in a heterogeneous system.

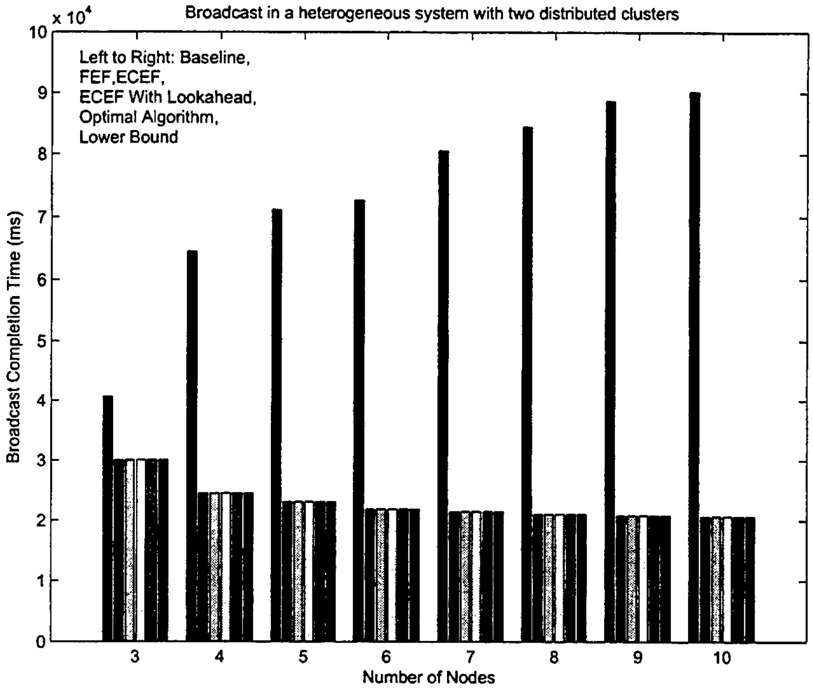


Figure 8: Simulation results for broadcast in a heterogeneous system with 2 distributed clusters.

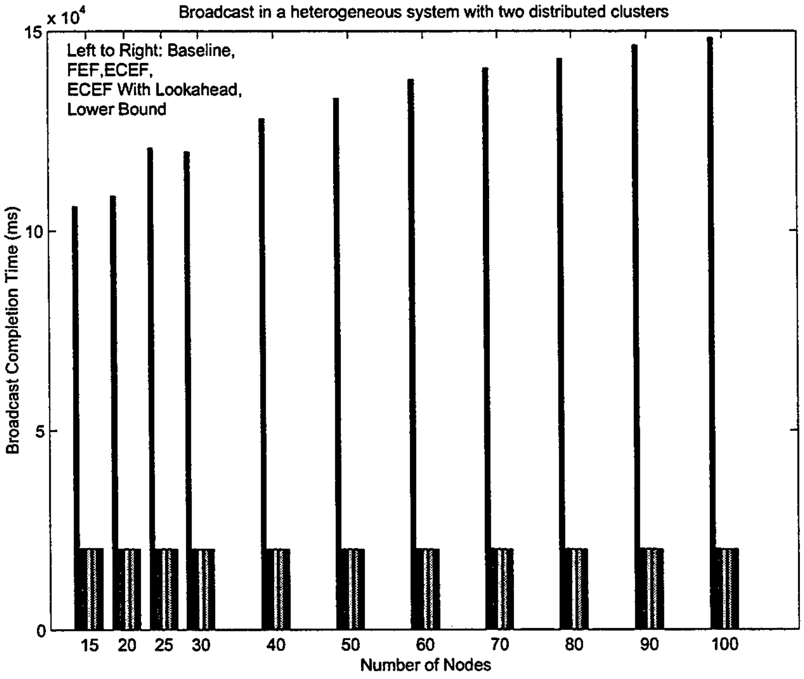


Figure 9: Simulation results for broadcast in a heterogeneous system with 2 distributed clusters.

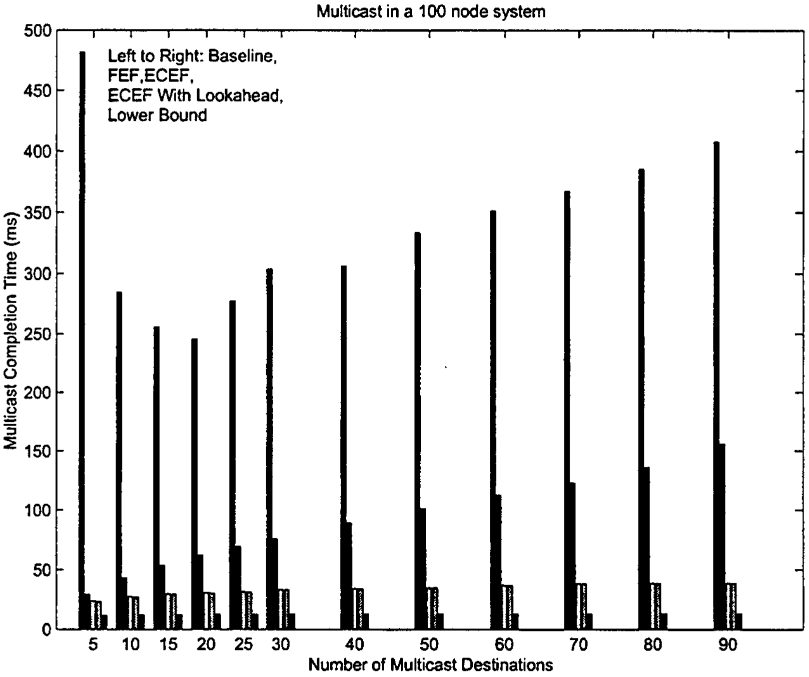


Figure 10: Simulation results for multicast.

Figure 1: A typical distributed heterogeneous system.
Figure 2: Broadcast schedules for the example in Eq (1): (a) Modified FNF schedule (b) Optimal schedule. The time values at each node represent the times at which the node is ready to send to another destination.
Figure 3: FEF communication schedule for the 4 node example of Eq (2).
Figure 4: Simulation results for broadcast in a heterogeneous system.
Figure 5: Simulation results for broadcast in a heterogeneous system.
Figure 6: Simulation results for broadcast in a heterogeneous system with 2 distributed clusters.
Figure 7: Simulation results for broadcast in a heterogeneous system with 2 distributed clusters.
Figure 8: Simulation results for multicast.

A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems

Tracy D. Braun[†], Howard Jay Siegel[†], Noah Beck[†], Ladislau L. Bölöni[‡],
Muthucumaru Maheswaran[§], Albert I. Reuther[†], James P. Robertson[*],
Mitchell D. Theys[**], Bin Yao[†], Debra Hensgen[◇], and Richard F. Freund[¶]

[†]School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285 USA
{tdbraun, hj, noah, reuther, yaob}@ecn.purdue.edu

[‡]Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
boloni@cs.purdue.edu

[§]Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2 Canada
maheswar@cs.umanitoba.ca

[*]Motorola
6300 Bridgepoint Parkway
Bldg. #3, MD: OE71
Austin, TX 78730 USA
robertso@ibmoto.com

[**]Department of Electrical Engineering
and Computer Science
University of Illinois at Chicago
Chicago, IL 60607-7053 USA
mtheys@eecs.uic.edu

[◇]Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118 USA
hensgen@cs.nps.navy.mil

[¶]NOEMIX
1425 Russ Blvd. Ste. T-110
San Diego, CA 92101 USA
rffreund@noemix.com

March 2000

Purdue University
School of Electrical and Computer Engineering
Technical Report TR-ECE 00-4

This research was supported in part by the DARPA/ITO Quorum Program under NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and under the GSA subcontract number GS09K99BH0250. Some of the equipment used was donated by Intel and Microsoft.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

ABSTRACT

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links to perform different computationally intensive applications that have diverse computational requirements. HC environments are well suited to meet the computational demands of large, diverse groups of tasks. The problem of mapping (defined as matching and scheduling) these tasks onto the machines of a distributed HC environment has been shown, in general, to be NP-complete, requiring the development of heuristic techniques. Selecting the best heuristic to use in a given environment, however, remains a difficult problem, because comparisons are often clouded by different underlying assumptions in the original studies of each heuristic. Therefore, a collection of eleven heuristics from the literature has been selected, adapted, implemented, and analyzed under one set of common assumptions. It is assumed that the heuristics derive a mapping statically (i.e., off-line). It is also assumed that a meta-task (i.e., a set of independent, non-communicating tasks) is being mapped, and that the goal is to minimize the total execution time of the meta-task. The eleven heuristics examined are Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-min, Max-min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A*. This study provides one even basis for comparison and insights into circumstances where one technique will out perform another. The evaluation procedure is specified, the heuristics are defined, and then comparison results are discussed. It is shown that for the cases studied here, the relatively simple Min-min heuristic performs well in comparison to the other techniques.

1. INTRODUCTION

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links to perform different computationally intensive applications that have diverse computational requirements [FrS93, MaB99, SiD97]. The general problem of mapping (i.e., matching and scheduling) tasks to machines in an HC suite has been shown to be NP-complete [Fer89, IbK77]. Heuristics developed to perform this mapping function are often difficult to compare because of different underlying assumptions in the original studies of each heuristic [BrS98]. Therefore, a collection of eleven heuristics from the literature has been selected, adapted, implemented, and compared by simulation studies under one set of common assumptions.

To facilitate these comparisons, certain simplifying assumptions were made. For these studies, let a meta-task be defined as a collection of independent tasks with no data dependencies (a given task, however, may have subtasks and dependencies among the subtasks). For this case study, it is assumed that static (i.e., off-line or predictive) mapping of meta-tasks is being performed. The goal of this mapping is to minimize the total execution time of the meta-task. Static mapping is useful for predictive analyzes (e.g., planning work for the next day), impact studies (e.g., determining the effect of purchasing another machine for the HC suite), and post-mortem analyzes (e.g., evaluating how well an on-line mapper performed).

It is also assumed that each machine executes a single task at a time (i.e., no multi-tasking), in the order in which the tasks are assigned. The size of the meta-task (i.e., the number of tasks to execute), $\underline{t}$, and the number of machines in the HC environment, $\underline{m}$, are static and known a *priori.*

This study provides one even basis for comparison and insights into circumstances where one mapping technique will out perform another. The evaluation procedure is specified, the heuristics are defined, and then comparison results are shown. It is shown that for the cases studied here, the relatively simple Min-min heuristic (defined in Chapter 3) performs well in comparison to other, more complex techniques investigated.

The remainder of this report is organized as follows. Chapter 2 defines the computational environment parameters that were varied in the simulations. Descriptions of the eleven mapping heuristics are found in Chapter 3. Chapter 4 examines selected results from the simulation study. A list of implementation parameters and procedures that could be varied for each heuristic is presented in Chapter 5.

This research was supported in part by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks) [HeK99]. MSHN is a collaborative research effort among the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver requested qualities of service. The heuristics developed in this paper or their derivatives may be included in the Scheduling Advisor component of the MSHN prototype.

2. SIMULATION MODEL

The eleven static mapping heuristics were evaluated using simulated execution times for an HC environment. Because these are static heuristics, it is assumed that an accurate estimate of the expected execution time for each task on each machine is known prior to execution and contained within an *ETC* (expected time to compute) matrix. One row of the *ETC* matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the *ETC* matrix consists of the estimated execution times of a given machine for each task in the meta-task. Thus, for an arbitrary task t_i and an arbitrary machine m_j, $ETC(t_i, m_j)$ is the estimated execution time of t_i on m_j.

For cases when inter-machine communications are required, $ETC(t_i, m_j)$ could be assumed to include the time to move the executables and data associated with task t_i from their known source to machine m_j. For cases when it is impossible to execute task t_i on machine m_j (e.g., if specialized hardware is needed), the value of $ETC(t_i, m_j)$ can be set to infinity, or some other arbitrary value. For this study, it is assumed that there are no inter-task communications, each task can execute on each machine, and the estimated expected execution times of each task on each machine are known. The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems (e.g., [GhY93, KaA98, SiY96]). (Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [KhP93, MaB99, SiD97].)

For the simulation studies, characteristics of the *ETC* matrices were varied in an attempt to represent a range of possible HC environments. The *ETC* matrices used were generated using the following method. Initially, a $t \times 1$ baseline column vector,

$\underline{B}$, of floating point values is created. Let $\underline{\phi_b}$ be the upper-bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $\underline{x_b^i} \in [1,\ \phi_b)$, and letting $B(i) = x_b^i$ for $0 \leq i < t$. Next, the rows of the *ETC* matrix are constructed. Each element $ETC(t_i, m_j)$ in row i of the *ETC* matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $\underline{x_r^{i,j}}$, which has an upper-bound of $\underline{\phi_r}$. This new random number, $x_r^{i,j} \in [1,\ \phi_r)$, is called a row multiplier. One row requires m different row multipliers, $0 \leq j < m$. Each row i of the *ETC* matrix can then be described as $ETC(t_i, m_j) = B(i) \times x_r^{i,j}$, for $0 \leq j < m$. (The baseline column itself does not appear in the final *ETC* matrix.) This process is repeated for each row until the $t \times m$ *ETC* matrix is full. Therefore, any given value in the *ETC* matrix is within the range $[1,\ \phi_b \times \phi_r)$ [MaA99].

To evaluate the heuristics for different mapping scenarios, the characteristics of the *ETC* matrix were varied based on several different methods from [Arm97]. The amount of variance among the execution times of tasks in the meta-task for a given machine is defined as task heterogeneity. Task heterogeneity was varied by changing the upper-bound of the random numbers within the baseline column vector. High task heterogeneity was represented by $\phi_b = 3000$ and low task heterogeneity used $\phi_b = 100$. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the upper-bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using $\phi_r = 1000$, while low machine heterogeneity values used $\phi_r = 10$. These heterogeneous ranges are based on one type of expected environment for MSHN. The ranges were chosen to reflect the fact that in real situations there is more variability across execution times for different tasks on a given machine than the execution time for a single task across different machines.

To further vary the ETC matrix in an attempt to capture more aspects of realistic mapping situations, different ETC matrix consistences were used. An ETC matrix is said to be consistent if whenever a machine m_j executes any task t_i faster than machine m_k, then machine m_j executes all tasks faster than machine m_k [Arm97]. Consistent matrices were generated by sorting each row of the ETC matrix independently, with machine m_0 always being the fastest and machine $m_{(m-1)}$ the slowest. In contrast, inconsistent matrices characterize the situation where machine m_j may be faster than machine m_k for some tasks, and slower for others. These matrices are left in the unordered, random state in which they were generated (i.e., no consistence is enforced). Partially-consistent matrices are inconsistent matrices that include a consistent submatrix. For the partially-consistent matrices used here, the row elements in column positions $\{0, 2, 4, \ldots\}$ of row i are extracted, sorted, and replaced in order, while the row elements in column positions $\{1, 3, 5, \ldots\}$ remain unordered (i.e., the even columns are consistent and the odd columns are, in general, inconsistent).

Sample ETC matrices for the twelve possible permutations of the characteristics listed above are shown in Tables 4.1 through 4.12. Results in this study used ETC matrices that had $t = 512$ tasks and $m = 16$ machines. These results (see Chapter 4) were taken as the average of 100 ETC matrices for each case.

While it was necessary to select some specific parameter values for t, m, and the ETC entries to allow implementation of a simulation, the techniques presented here are completely general. Therefore, if these parameter values do not apply to a specific situation of interest, researchers may substitute in their own values and the evaluation software of this study will still apply.

3. HEURISTIC DESCRIPTIONS

3.1 Introduction

The definitions of the eleven static meta-task mapping heuristics are provided below. First, some preliminary terms must be defined. Machine availability time, $mat(m_j)$, is the earliest time machine m_j can complete the execution of all the tasks that have previously been assigned to it. The completion time for a new task t_i on machine m_j, $ct(t_i, m_j)$, is the machine availability time plus the execution time of task t_i on machine m_j, i.e., $ct(t_i, m_j) = mat(m_j) + ETC(t_i, m_j)$. The performance criterion used to compare the results of the heuristics is the maximum value of $ct(t_i, m_j)$, for $0 \leq i < t$ and $0 \leq j < m$. The maximum $ct(t_i, m_j)$ value is also known as the makespan [Pin95]. Each heuristic is attempting to minimize the makespan (i.e., finish execution of the meta-task as soon as possible).

The descriptions below implicitly assume that the machine availability times are updated after each task is mapped. For heuristics where the tasks are considered in an arbitrary order, the order in which the tasks appeared in the ETC matrix was used. Most of the heuristics discussed here had to be adapted for this problem domain.

For many of the heuristics, there are control parameters values and/or control function specifications that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and/or information in the literature. These parameters and functions are mentioned in Chapter 5.

3.2 Heuristics

3.2.1 Opportunistic Load Balancing (OLB)

Opportunistic Load Balancing (OLB) assigns each task, in arbitrary order, to the next available machine, regardless of the task's expected execution time on that machine [ArH98, FrG98, FrS93]. The intuition behind OLB is to keep all machines as busy as possible. One advantage of OLB is its simplicity, but because OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans.

3.2.2 Minimum Execution Time (MET)

In contrast to OLB, Minimum Execution Time (MET) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine's availability [ArH98, FrG98]. The motivation behind MET is to give each task to its best machine. This can cause a severe load imbalance across machines. In general, this heuristic is obviously not applicable to HC environments characterized by consistent ETC matrices.

3.2.3 Minimum Completion Time (MCT)

Minimum Completion Time assigns each task, in arbitrary order, to the machine with the minimum completion time for that task [ArH98]. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The intuition behind MCT is to combine the benefits OLB and MET, while avoiding the circumstances in which OLB and MET perform poorly.

3.2.4 Min-min

The Min-min heuristic begins with the set $\underline{U}$ of all unmapped tasks. Then, the set of minimum completion times, $\underline{M} = \{\min_{0 \leq j < m}(ct(t_i, m_j))$, for each $t_i \in U\}$, is found. Next, the task with the overall *minimum* completion time from M is selected and assigned to the corresponding machine (hence the name Min-min). Lastly, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [ArH98, FrG98, IbK77]. Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time.

Min-min begins by scheduling the tasks that change the machine availability status by the least amount that any assignment could. For example, let t_i be the first task mapped by Min-min. The machine that finishes t_i the earliest, say m_j, is also the machine that executes t_i the fastest. For every task that Min-min maps after t_i, the Min-min heuristic changes the availability status of m_j by the least possible amount for every assignment. Therefore, the percentage of tasks assigned to their first choice (on the basis of execution time) is likely to be higher for Min-min than for Max-min (defined next). The expectation is that a smaller makespan can be obtained if more tasks are assigned to the machines that complete them the earliest and also execute them the fastest.

3.2.5 Max-min

The Max-min heuristic is very similar to Min-min. The Max-min heuristic also begins with the set $\underline{U}$ of all unmapped tasks. Then, the set of minimum completion times, M is found. Next, the task with the overall *maximum* completion time from M is selected and assigned to the corresponding machine (hence the name Max-min). Lastly, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [ArH98, FrG98, IbK77].

Intuitively, Max-min attempts to minimize the penalties incurred from performing tasks with longer execution times. Assume, for example, that the meta-task being mapped has many tasks with very short execution times, and one task with a very long execution time. Mapping the task with the longer execution time to its best machine first allows this task to be executed concurrently with the remaining tasks (with shorter execution times). For this case, this would be a better mapping than a Min-min mapping, where all of the shorter tasks would execute first, and then the longer running task would execute while several machines sit idle. Thus, in cases similar to this example, the Max-min heuristic may give a mapping with a more balanced load across machines and a better makespan.

3.2.6 Duplex

The Duplex heuristic is literally a combination of the Min-min and Max-min heuristics. The Duplex heuristic performs both of the Min-min and Max-min heuristics, and then uses the better solution [ArH98, FrG98]. Duplex can be performed to exploit the conditions in which either Min-min or Max-min performs well, with negligible overhead.

3.2.7 GA

Genetic Algorithms (GAs) have been studied for years [Hol75], and have become a popular technique used for searching large solution spaces (e.g., [SiY96, TiP96, WaS97]). The version of the heuristic used for this study was adapted from [WaS97] for this particular problem domain. Figure 3.1 shows the steps in a general GA.

The GA implemented here operates on a population of 200 chromosomes (possible mappings) for a given meta-task. Each chromosome is a $t \times 1$ vector, where position i ($0 \leq i < t$) represents task t_i, and the entry in position i is the machine to which the task has been mapped. The initial population is generated using two methods: (a) 200 randomly generated chromosomes from a uniform distribution, or (b) one chromosome that is the Min-min solution and 199 random solutions (mappings). The latter method is called seeding the population with a Min-min chromosome. The GA actually executes eight times (four times with initial populations from each method), and the best of the eight mappings is used as the final solution.

Each chromosome has a fitness value, which is the makespan that results from the matching of tasks to machines within that chromosome. After the generation of the initial population, all of the chromosomes in the population are evaluated based on their fitness value, with a smaller fitness value being a better mapping. Then, the main loop in Figure 3.1 is entered and a rank-based roulette wheel scheme [SrP94] is used for selection. This scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. Elitism, the property of guaranteeing the best solution remains in the population [Rud94], was also implemented. The population size stays fixed at 200.

Next, the crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. For the sections of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding tasks. Every chromosome is considered for crossover with a probability of 60%.

After crossover, the mutation operation is performed. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome, and randomly reassigns it to a new machine. Every chromosome is considered for mutation with a probability of 40%. For both crossover and mutation, the random operations select values from a uniform distribution.

Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when any one of three conditions are met: (a) 1000 total iterations, (b) no change in the elite chromosome for 150 iterations, or (c) all chromosomes converge to the same mapping. If no stopping criteria is met, the loop repeats, beginning with the selection of a new population. The stopping criteria that usually occurred in testing was no change in the elite chromosome in 150 iterations.

3.2.8 SA

Simulated Annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This solution uses the same representation for a solution as the chromosome for the GA. The initial implementation of SA was evaluated and then modified and refined to give a better final version. Both the initial and final implementations are described below.

SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [CoP96, KiG83, RuN95, ZoK99]). This probability is based on a system temperature that decreases for each iteration. As the system temperature "cools," it is more difficult for poorer solutions to be accepted. The initial system temperature is the makespan of the initial (random) mapping.

The initial SA procedure implemented here is as follows. The first mapping is generated from a uniform random distribution. The mapping is mutated in the same manner as the GA, and the new makespan is evaluated. The decision algorithm for accepting or rejecting the new mapping is based on [CoP96]. If the new makespan is better, the new mapping replaces the old one. If the new makespan is worse (larger), a uniform random number $z \in [0,\ 1)$ is selected. Then, z is compared with y, where

$$y = \frac{1}{1 + e^{\left(\frac{\text{old makespan}-\text{new makespan}}{\text{temperature}}\right)}}. \tag{3.1}$$

If $z > y$ the new (poorer) mapping is accepted, otherwise it is rejected, and the old mapping is kept.

Notice that for solutions with similar makespans (or if the system temperature is very large), $y \to 0.5$, and poorer solutions are accepted with approximately a 50% probability. In contrast, for solutions with very different makespans (or if the system temperature is very small), $y \to 1$, and poorer solutions will usually be rejected.

After each mutation, the system temperature is reduced to 90% of its original value. (This percentage is defined as the cooling rate.) This completes one iteration of SA. The heuristic stops when there is no change in the makespan for 150 iterations or the system temperature approaches zero. Most tests ended when the system temperature approached zero (approximated by 10^{-200}).

Results from preliminary studies using the initial implementation described above showed that the GA usually found the best mappings of all eleven heuristics. However, the execution time of the SA heuristic was much shorter than that of the GA. Therefore, in order to try and provide a more "fair" comparison, the SA heuristic was adapted so that it would be more similar to GA. The modifications gave SA an execution time as long as GA. The longer execution time allowed more of the solution space to be searched with the SA procedure, with the hope that SA would then find more competitive mappings.

To try to make SA more competitive with GA, the following changes were made to the the final SA implementation. First, the stopping conditions were modified. The number of unchanged iterations was raised to 200 and two different cooling rates were used, 80% and 90%. Next, SA was allowed to execute eight times for each cooling rate, using the best solution from all sixteen runs as the final mapping. Lastly, four of the eight runs for each cooling rate were seeded with the Min-min solution, just as with the GA.

Even with the additional execution time and Min-min seedings, SA still found poorer solutions than Min-min or GA. Because SA allows poorer solutions to be accepted at intermediate stages, it allows some very poor solutions in the initial stages, from which it can never recover (see Chapter 4).

3.2.9 GSA

The Genetic Simulated Annealing (GSA) heuristic is a combination of the GA and SA techniques [ChF98, ShW96]. In general, GSA follows procedures similar to the GA outlined above. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simplified SA decision process for accepting or rejecting a new chromosomes.

Specifically, the initial system temperature was set to the average makespan of the initial population, and decreased 10% for each iteration. When a new (post-mutation, post-crossover, or both) chromosome is compared with the corresponding original chromosome, if the new makespan is less than the original makespan plus the system temperature, then the new chromosome is accepted. Otherwise, the original chromosome survives to the next iteration. Therefore, as the system temperature decreases, it is again more difficult for poorer solutions to be accepted. The two stopping criteria used were either (a) no change in the elite chromosome in 150 iterations or (b) 1000 total iterations. The most common stopping criteria was no change in the elite chromosome in 150 iterations.

3.2.10 Tabu

Tabu search is a solution space search that keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas [DeD94, GlL97]. A solution (mapping) uses the same representation as a chromosome in the GA approach.

The implementation of Tabu search used here begins with a random mapping, generated from a uniform distribution. To manipulate the current solution and move through the solution space, a short hop is performed. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. The basic procedure for performing a short hop is to select a pair of tasks and assign them to every possible combination of machines. This is done for every possible pair of tasks. The pseudocode for the short hop procedure is given in Figure 3.2.

Let the tasks in the pair under consideration be denoted ti and tj in Figure 3.2. (The machine assignments for the other $t-2$ tasks are held fixed.) The machines to which tasks ti and tj are remapped are mi and mj, respectively. For each possible pair of tasks, each possible pair of machine assignments is considered. Lines 1 through 4 set the boundary values of the different loops. Line 6 or 8 is where each new solution (mapping) is evaluated, and line 9 is where the new solution is considered for acceptance. Each of these new solutions is a short hop. If the new makespan is an improvement, the new solution is saved, replacing the current solution. (This is defined as a successful short hop.) When ti and tj represent the same task (ti = tj), a special case occurs (line 5). In these situations, all machines for that one task are considered.

When any new solution is found to be an improvement (line 10), the procedure breaks out of the for loops, and starts searching from the beginning again. The short hop procedure ends when (1) every pair-wise remapping combination has been exhausted with no improvement (i.e., the bounds of all four for loops in Figure 3.2 have been reached), or (2) the limit on the total number of successful hops, $limit_{hops}$ is reached.

When the short hop procedure ends, the final mapping from the local solution space search is added to the tabu list. The tabu list is a method of keeping track of the regions of the solution space that have already been searched. Next, a new random mapping is generated, and it must differ from each mapping in the tabu list by at least half of the machine assignments (a successful long hop). The intuitive purpose of a long hop is to move to a new region of the solution space that has not already been searched.

The final stopping criterion for the heuristic is determined by the total number of successful long and short hops combined. That is, when the sum of the total number of successful short hops and successful long hops equals $limit_{hops}$, the heuristic ends. Then, the best mapping from the tabu list is the final answer.

Similar to SA, some parameters of Tabu were varied in an attempt to make Tabu more competitive with GA, while also trying to provide a more "fair" comparison between Tabu and GA. To this end, the value used for $limit_{hops}$ was varied depending on the type of consistency of the matrix being considered.

Because of the implementation of the short hop procedure described above, the execution time of the Tabu search depended greatly on the type of consistency of the ETC matrix. Each time a new task is considered for remapping in the short hop procedure, it is first considered on m_0, then m_1, etc. For consistent matrices, these will be the fastest machines. Therefore, once a task gets reassigned to a fast machine, the remaining permutations of the short hop procedure will be unsuccessful. In other words, because the short hop procedure begins searching sequentially from the best machines, there will be a larger number of unsuccessful hops performed for each successful hop for consistent ETC matrices. Thus, the execution time of Tabu will increase.

Therefore, to keep execution times "fair" and competitive with GA, $limit_{hops}$ was set to 1000 for consistent ETC matrices, 2000 for partially-consistent matrices, and 2500 for inconsistent matrices. When most test cases had stopped, the percentage of successful short hops was high (90% or more) relative to the percentage of successful long hops (10% or less). But because there were long hops being performed, every pairwise combination of short hops was being exhausted, and new, different regions of the solution space were being searched.

3.2.11 A*

The final heuristic in the comparison study is known as the A* heuristic. A* has been applied to many other task allocation problems (e.g., [ChL91, KaA98, RuN95, ShT85]). The technique used here is similar to [ChL91].

A* is a search technique based on an m-ary tree, beginning at a root node that is a null solution. As the tree grows, nodes represent partial mappings (a subset of tasks are assigned to machines). The partial mapping (solution) of a child node has one more task mapped than the parent node. Call this additional task $\underline{a}$. Each parent node generates m children, one for each possible mapping of a. After a parent node has done this, the parent node becomes inactive. To keep execution time of the heuristic tractable, there is a pruning process to limit the maximum number of active nodes in the tree at any one time (in this study, to 1024).

Each node, $\underline{n}$, has a cost function, $\underline{f(n)}$, associated with it. The cost function is an estimated lower-bound on the makespan of the best solution that includes the partial solution represented by node n.

Let $\underline{g(n)}$ represent the makespan of the task/machine assignments in the partial solution of node n, i.e., $g(n)$ is the maximum of the machine availability times ($mat(m_j)$) based on the set of tasks that have been mapped to machines in node n's partial solution. Let $\underline{h(n)}$ be a lower-bound estimate on the difference between the makespan of node n's partial solution and the makespan for the best complete solution that includes node n's partial solution. Then, the cost function for node n is computed as

$$f(n) = g(n) + h(n). \tag{3.2}$$

Therefore, $f(n)$ represents the makespan of the partial solution of node n plus a lower-bound estimate of the time to execute the rest of the (unmapped) tasks in the meta-task.

The function $h(n)$ is defined in terms of two functions, $\underline{h_1(n)}$ and $\underline{h_2(n)}$, which are two different approaches to deriving a lower-bound estimate. Recall that $M = \{\min_{0 \le j < m}(ct(t_i, m_j)),$ for each $t_i \in U\}$. For node n let $\underline{mmct(n)}$ be the overall maximum element of M (i.e., "the maximum minimum completion time"). Intuitively, $mmct(n)$ represents the best possible meta-task makespan by making the typically unrealistic assumption that each task in U can be assigned to the machine indicated in M without conflict. Thus, based on [ChL91], $h_1(n)$ is defined as

$$h_1(n) = \max(0,\ (mmct(n) - g(n))). \tag{3.3}$$

Next, let $\underline{sdma(n)}$ be the sum of the differences between $g(n)$ and each machine availability time over all machines after executing all of the tasks in the partial solution represented by node n:

$$sdma(n) = \sum_{j=0}^{m-1} (g(n) - mat(m_j)). \tag{3.4}$$

Intuitively, $sdma(n)$ represents the collective amount of machine availability time remaining that can be scheduled without increasing the final makespan. Let $\underline{smet(n)}$ be defined as the sum over all tasks in U of the minimum expected execution time (i.e., ETC value) for each task in U:

$$smet(n) = \sum_{t_i \in U} (\min_{0 \leq j < m} (ETC(t_i, m_j)) \tag{3.5}$$

This gives an estimate of the amount of remaining work to do, which could increase the final makespan. The function h_2 is then defined as

$$h_2(n) = \max(0,\ (smet(n) - sdma(n))/m), \tag{3.6}$$

where $(smet(n) - sdma(n))/m$ represents an estimate of the minimum increase in the meta-task makespan if the tasks in U could be "ideally" (but, in general, unrealistically) distributed among the machines. Using these definitions,

$$h(n) = \max(h_1(n), h_2(n)), \tag{3.7}$$

representing a lower-bound estimate on the time to execute the tasks in U.

Thus, after the root node generates m nodes for t_0 (each node mapping t_0 to a different machine), the node with the minimum $f(n)$ generates its m children, until 1024 nodes are created. From that point on, any time a node is added, the tree is pruned by deactivating the leaf node with the largest $f(n)$. This process continues until a leaf node representing a complete mapping is reached. Note that if the tree is not pruned, this method is equivalent to an exhaustive search.

3.3 Concluding Remarks

This set of eleven static mapping heuristics is not exhaustive, nor is it meant to be. It is simply a representative set of several different approaches, including iterative, non-iterative, greedy, and biologically inspired techniques. Several other types of static mapping heuristics exist. For example, other techniques that have been or could be used as static mappers for heterogeneous computing environments include the following: neural networks [ChH98], linear programming [CoL92], the "Mapping Heuristic" (MH) algorithm [ElL90], the Cluster-M technique [EsW96], the Levelized Min Time (LMT) algorithm [IvO95], the *k*-percent best (KPB) and Sufferage heuristics [MaA99], the Dynamic Level Scheduling (DLS) algorithm [SiL93], recursive bisection [SiT97], and the Heterogeneous Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CROP) techniques [ToH99]. The eleven heuristics examined here were initially selected because they seemed among the most appropriate for the static mapping of meta-tasks, and covered a wide range of techniques.

4. EXPERIMENTAL RESULTS

4.1 Introduction

An interactive software application has been developed that allows simulation, testing, and demonstration of the heuristics examined in Chapter 3, applied to the meta-tasks defined by the *ETC* matrices described in Chapter 2. The software allows a user to specify t and m, to select which type of *ETC* matrices to use, and to choose which heuristics to execute. It then generates the specified *ETC* matrices, executes the desired heuristics, and displays the results, similar to Figures 4.1 through 4.12. The results discussed in this chapter were generated using this software.

4.2 Results for 512 Tasks

4.2.1 Heuristic Execution Times

When comparing mapping heuristics, the execution time of the heuristics themselves is an important consideration. For the eleven heuristics that were compared, the execution times varied greatly. The experimental results discussed below were obtained on a Pentium II 400 MHz processor with 1GB of RAM. The heuristic execution times are the average time each heuristic took to compute a mapping for a single 512 task × 16 machine *ETC* matrix, averaged over 100 different matrices (each of the same type).

The first three heuristics described, OLB, MET, and MCT, each of which has asymptotic complexity of $O(mt)$, executed in less than one microsecond per *ETC* matrix. Next, the Min-min, Max-min, and Duplex heuristics, each with asymptotic complexity $O(mt^2)$, executed in an average of 200 milliseconds. The GA, which usually provided the best results (in terms of makespan), had an average execution time of 60 seconds. GSA, which uses many procedures similar to the GA, had an average execution time of 69 seconds. As described in the previous chapter, SA and Tabu were adapted to provide a more fair comparison with the results of the GA, so their average execution times were also approximately 60 seconds per *ETC* matrix. Finally A*, which has exponential complexity, executed in an average of over 20 minutes (1200 seconds).

The resulting makespans (i.e., the time it would take for a given meta-task to complete on the heterogeneous environment) from the simulations for every case of consistency, task heterogeneity, and machine heterogeneity are shown in Figures 4.1 through 4.12. After each figure is a table with a sample 8 × 8 subsection from one of the 512 × 16 *ETC* matrices with the same type of consistency (Tables 4.1 through 4.12). All experimental results represent the average makespan for a meta-task of the defined type of *ETC* matrix. For each heuristic and each type of *ETC* matrix, the results were averaged over 100 different *ETC* matrices of the same type (i.e., 100 mappings). The range bars for each heuristic show the 95% confidence interval [Jai91] (min, max) for the average makespan. This interval represents the likelihood that makespans of mappings for that type of heterogeneity and heuristic fall within the specified range. That is, if another ETC matrix (of the same type) was generated, and the specified heuristic generated a mapping, then the makespan of the mapping would be within the given confidence interval with 95% certainty.

4.2.2 Consistent Heterogeneity

The results for the meta-task execution times for the four consistent cases are shown in Figures 4.1, 4.2, 4.3, and 4.4. The corresponding *ETC* matrix excerpts are in Tables 4.1, 4.2, 4.3, and 4.4. The differences in magnitude on the y-axis among the graphs are from the different ranges of magnitude used in generating the different types of *ETC* matrices.

For both cases of low machine heterogeneity, the relative performance order of the heuristics from best to worst was: (1) GA, (2) Min-min, (3) Duplex, (4) GSA, (5) A*, (6) Tabu, (7) MCT, (8) SA, (9) Max-min, (10) OLB, and (11) MET. For the two high machine heterogeneity cases, the relative performance order of the heuristics from best to worst was: (1) GA, (2) Min-min, (3) Duplex, (4) A*, (5) GSA, (6) MCT, (7) Tabu, (8) SA, (9) Max-min, (10) OLB, and (11) MET. For consistent ETC matrices, the MET algorithm mapped all tasks to the same machine, resulting in the worst performance by an order of magnitude. Therefore, MET is not included in the figures for the consistent cases. The performance of the heuristics will be discussed in the order in which they appear in the figures.

For all four consistent cases, OLB gave the second worst results (after MET). In OLB, the first m tasks get assigned, one each, to the m idle machines. Because of the the consistent ETC matrix, there will be some very poor initial mappings (tasks $m-2$ and $m-1$, for example, get their worst machines). Because task execution times are not considered, OLB may continue to assign tasks to machines where they execute slowly, hence the poor makespans for OLB.

MCT always performed around the median of the heuristics, giving the sixth best (low machine heterogeneity) or seventh best (high machine heterogeneity) results. MCT only makes one iteration through the ETC matrix, assigning tasks in the order in which they appear in the ETC matrix, hence it can only make mapping decisions of limited scope, and it cannot make globally intelligent decisions like Min-min or A*.

The Min-min heuristic performed very well for consistent ETC matrices, giving the second best result in each case. Not only did Min-min always give the second best mapping, but the Min-min mapping was always within ten percent of the best mapping found (which was with GA, discussed below). Min-min is able to make globally intelligent decisions to minimize task completion times, which also results in good machine utilization and good makespans. Similar arguments hold for the Duplex heuristic.

In contrast, the Max-min heuristic always performed poorly, giving only the ninth best mapping. Consider the state of the machine ready times during the execution of the Min-min and Max-min heuristics. Min-min always makes the assignment that changes the machine ready times by the least amount. In general, the assignment made by Max-min will change the machine ready times by a larger amount. Therefore, the values of the machine ready times for each machine will remain closer to each other when using the Min-min heuristic than when using the Max-min heuristic. Both Min-min and Max-min will assign a given task to the machine that gives the best *completion* time. However, if the machine ready times remain close to each other, then Min-min gives each task a better chance of being assigned to the machine that gives the task its best *execution* time. In contrast, with Max-min, there is a higher probability of there being relatively greater differences among the machine ready times. This results in a "load balancing" effect, and each task has a lower chance of being assigned to the machine that gives the task its best *execution* time.

For the heterogeneous environments considered in this study, the type of special case where Max-min may outperform Min-min (as discussed in Chapter 3) never occurs. Min-min found a better mapping than Max-min every time (i.e., in each of the 100 trials for each type of heterogeneity). Thus, Max-min performed poorly in this study. As a direct result, the Duplex heuristic always selected the Min-min solution, giving Duplex a tie for the second best solution. (Because Duplex always relied on the Min-min solution, it is listed in third place.)

GA provided the best mappings for the consistent cases. This was due in large part to the good performance of the Min-min heuristic. The best GA solution always came from one of the populations that had been seeded with the Min-min solution. However, the additional searching capabilities afforded to GA by performing crossover and mutation were beneficial, as the GA was always able to improve upon this solution by five to ten percent.

SA, which manipulates a single solution, ranked eighth for both types of machine heterogeneity. For this type of heterogeneous environment, this heuristic (as implemented here) do not perform as well as the GA which had similar execution time and Min-min which had a faster execution time. While the SA procedure is iterative (like the GA procedure), it appears that the crossover operation and selection procedure of the GA are advantageous for this problem domain.

The mapping found by GSA was either the fourth best (low machine heterogeneity) or the fifth best (high machine heterogeneity) mapping found, alternating with A*. GSA does well for reasons similar to those described for GA. The average makespan found by GSA could have been slightly better, but the results were hindered by a few very poor mappings that were found. These very poor mappings result in the large confidence intervals found in the figures for GSA. Thus, for these heterogeneous environments, the selection method from GA does better than the method from GSA.

Tabu provides fairly constant results, always finding the sixth or seventh best mapping (alternating with MCT). As noted in the previous chapter, because of the short hop procedure implemented and the structure of the consistent matrices, Tabu finds most of the successful short hops right away and must then perform a large number of unsuccessful short hops (recall machine m_i outperforms machine m_{i+1} for the consistent cases). Because the stopping criteria is determined by the number of successful hops, and because each short hop procedure has few successful hops, more successful long hops are generated, and more of the solution space is searched. Thus, Tabu performs better for consistent matrices than for inconsistent.

Considering the order of magnitude difference in execution times between A* and the other heuristics, the quality of the mappings found by A* was disappointing. The A* mappings alternated between fourth and fifth best with GSA. The performance of A* was hindered because the estimates made by $h_1(n)$ and $h_2(n)$ are not as accurate for consistent cases as they are for inconsistent and partially-consistent cases.

For consistent cases, $h_1(n)$ underestimates the competition for machines and $h_2(n)$ overestimates the number of tasks that can be assigned to their best machine.

4.2.3 Inconsistent Heterogeneity

For the four inconsistent test cases in Figures 4.5 through 4.8 and Tables 4.5 through 4.8, one sees similar trends in all four cases. For both cases of low machine heterogeneity, the relative performance order of the heuristics from best to worst was: (1) GA, (2) A*, (3) Min-min, (4) Duplex, (5) MCT, (6) MET, (7) GSA, (8) SA, (9) Tabu, (10) Max-min, and (11) OLB. For the two high machine heterogeneity cases, the relative performance order of the heuristics from best to worst was: (1) GA, (2) A*, (3) Min-min, (4) Duplex, (5) MCT, (6) MET, (7) SA, (8) GSA, (9) Max-min, (10) Tabu, and (11) OLB.

MET performs much better than in the consistent cases, while the performance of OLB degrades. The reason OLB does better for consistent than inconsistent matrices is as follows. Consider for example, machine m_0 and machine m_1 in the consistent case. By definition, all tasks assigned to m_0 will be on their best machine, and all tasks assigned to m_1 will be on their second best machine. However, OLB ignores direct consideration of the execution times of tasks on machines. Thus, for the inconsistent case, none of the tasks assigned to m_0 may be on their best machine, and none of the tasks assigned to m_1 may be on their second best machine, etc. Therefore, it is more likely that OLB will assign more tasks to poor machines, resulting in the worst mappings for each of the inconsistent cases. In contrast, MET improves and finds the sixth best schedules because the "best" machines are distributed across the set of machines, thus task assignments will be more evenly distributed among the set of machines avoiding load imbalance.

Similarly, MCT can also exploit the fact that the machines providing the best task completion times are more evenly distributed among the set of machines. Thus, by assigning each task, in the order specified by the ETC matrix, to the machine that completes it the soonest, there is a better chance of assigning a task to a machine that executes it well, decreasing the overall makespan.

Min-min continued to give better results than Max-min (which ranked ninth or tenth), by a factor of about two for all of the inconsistent cases. In fact, Min-min was again one of the best of all eleven heuristics, giving the third best mappings, which produced makespans that were still within 12% of the best makespans found. As noted earlier, Duplex selected the Min-min solution in every case, and so ranked fourth.

GA provided the best mappings for the inconsistent cases. GA was again able to benefit from the performance of Min-min, as the best solution always came from from one of the populations seeded with the Min-min solution. GA has provided the best solution in all consistent and inconsistent cases examined, and its execution time is largely independent of any of the heterogeneity characteristics. This makes it a good general-purpose heuristic, when mapper execution time is not a critical issue.

SA and GSA had similar results, alternating between the seventh and eighth best schedules. For the high machine heterogeneity cases, SA found mappings that were better by about 25%. For the low machine heterogeneity cases, GSA found the better mappings, but only by 3 to 11%.

Tabu performs very poorly (ninth or tenth best) for inconsistent matrices when compared to its performance for consistent matrices (sixth or seventh best). The sequential procedure for generating short hops, combined with the inconsistent structure of the *ETC* matrices, results in Tabu finding more successful short hops, and performing fewer unsuccessful short hops. Many more intermediate solutions of marginal improvement exist within an inconsistent *ETC* matrix. Therefore, the hop limit is reached faster because of all the successful short hops (even though the hop limit is higher). Thus, less of the solution space is searched, and the result is a poor solution. That is, for the inconsistent case, the ratio of successful short hops to successful long hops increases, as compared to the consistent case, and fewer areas in the search space are examined.

A* had the second best average makespans, behind GA, and both of these methods produced results that were usually within a small factor of each other. A* did well because if the machines with the fastest execution times for different tasks are more evenly distributed, the lower-bound estimates of $h_1(n)$ and $h_2(n)$ are more accurate.

4.2.4 Partially-consistent Heterogeneity

Finally, consider the partially-consistent cases in Figures 4.9 through 4.12 and Tables 4.9 through 4.12. For both cases of low machine heterogeneity, the relative performance order of the heuristics from best to worst was: (1) GA, (2) Min-min, (3) Duplex, (4) A*, (5) MCT, (6) GSA, (7) Tabu, (8) SA, (9) Max-min, (10) OLB, and (11) MET. For the high task, high machine heterogeneity cases, the relative performance order of the heuristics from best to worst was: (1) GA, (2) Min-min, (3) Duplex, (4) A*, (5) MCT, (6) GSA, (7) SA, (8) Tabu, (9) Max-min, (10) OLB, and (11) MET. The rankings for low task, high machine heterogeneity were similar to high task, high machine heterogeneity, except GSA and SA are switched in order.

The MET performed the worst for every partially-consistent case. Intuitively, MET is suffering from the same problem as in the consistent cases: half of all tasks are getting assigned to the same machine.

OLB does poorly for high machine heterogeneity cases because bad assignments will have higher execution times for high machine heterogeneity. For low machine heterogeneity, the bad assignments have a much lower penalty. In all four cases, OLB was the second worst approach.

MCT again performs relatively well (fifth best) because the machines providing the best task completion times are more evenly distributed among the set of machines, similar to the inconsistent cases. Max-min continued to do poorly and ranked ninth.

The Duplex solutions were the same as the Min-min solutions, and tied for second best. The rankings for SA, GSA, and Tabu were approximately the averages of what they were for the consistent and inconsistent cases, as might be expected.

The best heuristics for the partially-consistent cases were GA (best), and Min-min (second best), followed closely by A* (fourth best, after Duplex). This is not surprising because these were among the best heuristics from the consistent and inconsistent tests, and partially-consistent matrices are a combination of consistent and inconsistent matrices. Min-min was able to do well because it's approach assigned a high percentage of tasks to their first choice of machines. A* was robust enough to handle the consistent components of the matrices, and did well for the same reasons mentioned for inconsistent matrices. GA maintained its position as best heuristic. The execution time and performance of GA is largely independent of heterogeneity characteristics. The additional regions of the solution space that are searched by the GA mutation and crossover operations are beneficial, as they were always able to improve on the Min-min solution by five to ten percent.

4.3 Summary

To summarize the findings of this chapter, for consistent *ETC* matrices, GA gave the best results, Min-min the second best, and MET gave the worst. When the *ETC* matrices were inconsistent, OLB provided the poorest mappings while the mappings from GA and A* performed the best. For the partially-consistent cases, GA still gave the best results, followed closely by Min-min and A*, while MET had the slowest. All results were for meta-tasks with $t = 512$ tasks executing on $m = 16$ machines, averaged over 100 different trials.

For the situations considered in this study, the relative performance of the mapping heuristics varied based on the characteristics of the HC environments. The GA always gave the best performance. If mapper execution time is also considered, Min-min gave excellent performance (within 12% of the best) and had a very small execution time.

5. ALTERNATIVE IMPLEMENTATIONS

The experimental results in Chapter 4 show the performance of each heuristic under the assumptions presented. For several heuristics, specific control parameter values and control functions had to be selected. In most cases, control parameter values and control functions were based on the references cited and/or preliminary experiments that were conducted. However, for these heuristics, several different, valid implementations are possible using different control parameters and control functions. Some of these control parameters and control functions are listed below for selected heuristics.

GA: Several control parameter values could be varied in the GA, including population size, crossover probability, mutation probability, stopping criteria, and number of initial populations considered per result. Specific functions within GA controlling the progress of the search that could be changed are initial population "seed" generation, mutation, crossover, selection, and elitism.

SA: Parameter values with SA that could be modified are system temperature, cooling rate, stopping criteria, and the number of runs per result. Adaptable control procedures in SA include the initial population "seed" generation, mutation, and the equation for deciding when to accept a poorer solution.

GSA: Like the two heuristics its based upon, GSA also has several parameters that could be varied, including: population size, crossover probability, mutation probability, stopping criteria, cooling rate, number of runs with different initial populations per result, and the system temperature. The specific procedures used for the following actions could also be modified: initial population "seed" generation, mutation, crossover, selection, and the equation for deciding when to accept a poorer solution.

Tabu: The short hop method implemented was a "first descent" (take the first improvement possible) method. "Steepest descent" methods (where several short hops are considered simultaneously, and the one with the most improvement is selected) are also used in practice [DeD94]. Other techniques that could be varied are the long hop method, the order of the short hop pair generation-and-exchange sequence, and the stopping condition. Two possible alternative stopping criteria are when the tabu list reaches a specified number of entries, or when there is no change in the best solution in a specified number of hops.

A*: Several variations of the A* method that was employed here could be implemented. Different functions could be used to estimate the lower-bound $h(n)$. The maximum size of the search tree could be varied, and several other techniques exist for tree pruning (e.g., [RuN95]).

In summary, for the GA, SA, GSA, Tabu, and A* heuristics there are a great number of possible valid implementations. An attempt was made to use a reasonable implementation of each heuristic for this study. Future work could examine other implementations.

6. CONCLUSIONS

The goal of this study was to provide a basis for comparison and insights into circumstances where one technique will out perform another for eleven different heuristics. The characteristics of the *ETC* matrices used as input for the heuristics and the methods used to generate them were specified. The implementation of a collection of eleven heuristics from the literature was described. The results of the mapping heuristics were discussed, revealing the best heuristics to use in certain scenarios. For the situations, implementations, and parameter values used here, GA consistently gave the best results. The average performance of the relatively simple Min-min heuristic was always within twelve percent of the GA heuristic.

The comparisons of the eleven heuristics and twelve situations provided in this study can be used by researchers as a starting point when choosing heuristics to apply in different scenarios. They can also be used by researchers for selecting heuristics to compare new, developing techniques against.

Acknowledgments – The authors thank Shoukat Ali for his comments. Portions of this document appear in the proceedings of the *8th IEEE Workshop on Heterogeneous Computing (HCW '99).*

LIST OF REFERENCES

[ArH98] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79–87.

[Arm97] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Sept. 1997 (D. Hensgen, advisor.)

[BrS98] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *IEEE Workshop on Advances in Parallel and Distributed Systems (APADS '98)*, Oct. 1998, pp. 330–335 (included in *17th IEEE Symposium on Reliable Distributed Systems*, 1998).

[ChF98] H. Chen, N. S. Flann, and D. W. Watson, "Parallel genetic simulated annealing: A massively parallel SIMD approach," *IEEE Transactions on Parallel and Distributed Computing*, Vol. 9, No. 2, Feb. 1998, pp. 126–136.

[ChH98] R.-M. Chen and Y.-M. Huang, "Multiconstraint task scheduling in multiprocessor systems by neural networks," *10th IEEE Conference on Tools with Artificial Intelligence*, Nov. 1998, pp. 288–294.

[ChL91] K. Chow and B. Liu, "On mapping signal processing algorithms to a heterogeneous multiprocessor system," *1991 International Conference on Acoustics, Speech, and Signal Processing - ICASSP 91*, Vol. 3, May 1991, pp. 1585–1588.

[CoP96] M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, Vol. 42, No. 6-7, Dec. 1996, pp. 465–475.

[CoL92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1992.

[DeD94] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro, "Improving search by incorporating evolution principles in parallel tabu search," *1994 IEEE Conference on Evolutionary Computation*, Vol. 2, 1994, pp. 823–828.

[ElL90] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of Parallel and Distributed Computing*, Vol. 9, No. 2, June 1990, pp. 138–153.

[EsW96] M. M. Eshaghian and Y.-C. Wu, "Mapping and resource estimation in network heterogeneous computing," in *Heterogeneous Computing*, M. M. Eshaghian, ed., Artech House, Boston, MA, 1996, pp. 197–223.

[Fer89] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[FrG98] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184–199.

[FrS93] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13–17.

[GhY93] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[GlL97] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, MA, June 1997.

[HeK99] D. A. Hensgen, T. Kidd, M. C. Schnaidt, D. St. John, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J-K. Kim, C. Irvine, T. Levin, R. Wright, R. F. Freund, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: A management system for heterogeneous networks," *8th IEEE Workshop on Heterogeneous Computing Systems (HCW '99)*, Apr. 1999, pp. 184–198.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[IbK77] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[IvO95] M. Iverson, F. Ozguner, G. Follen, "Parallelizing existing application in a distributed heterogeneous environment," *4th IEEE Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 93–100.

[Jai91] R. Jain, *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley & Sons, New York, NY, 1991.

[KaA98] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July–Sept. 1998, pp. 42–51.

[KhP93] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18–27.

[KiG83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, Vol. 220, No. 4598, May 1983, pp. 671–680.

[MaA99] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–121

[MaB99] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., John Wiley & Sons, New York, NY, 1999, Vol. 8, pp. 679–690.

[Pin95] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[Rud94] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, Jan. 1994, pp. 96–101.

[RuN95] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[ShT85] C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing system using a minimax criterion," *IEEE Transactions on Computers*, Vol. C-34, No. 3, Mar. 1985, pp. 197–203.

[ShW96] P. Shroff, D. Watson, N. Flann, and R. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, April 1996, pp. 98–104.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.

[SiL93] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, Feb. 1993, pp. 175–186.

[SiT97] H. D. Simon and S.-H. Teng, "How good is recursive bisection?" *SIAM Journal on Scientific Computing*, Vol. 18, No. 5, Sept. 1997, pp. 1436–1445.

[SiY96] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[SrP94] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17–26.

[TiP96] Y. G. Tirat-Gefen and A. C. Parker, "MEGA: An approach to system-level design of application specific heterogeneous multiprocessors," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 105–117.

[ToH99] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 3–14.

[WaS97] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 1–15.

[ZoK99] A. Y. Zomaya and R. Kazman, "Simulated annealing techniques," *Algorithms and Theory of Computation Handbook*, M. J. Atallah, ed., CRC Press, Boca Raton, FL, 1999, pp. 37-1–37-19.

```
initial population generation;
evaluation;
while (stopping criteria not met) {
    selection;
    crossover;
    mutation;
    evaluation;
}
```

Figure 3.1. General procedure for a Genetic Algorithm, based on [SrP94].

```
LOOP:  /* begin short hop procedure */
       for ti = 0 to $t-1$        /* first task in pair */
           for mi = 0 to $m-1$        /* first machine in pair */
               for tj = ti to $t-1$        /* second task in pair */
                   for mj = 0 to $m-1$        /* second machine in pair */

                       if (ti == tj)
                           evaluate new solution
                           with task tj on machine mj;
                       else
                           evaluate new solution with
                           task ti on machine mi and
                           task tj on machine mj;

                       if (new solution is better) {
                           replace old solution with new solution;
                           successful_hops = successful_hops + 1;
                           goto LOOP;        /* restart from inital state */
                       }

                       if (successful_hops == $limit_{hops}$)
                           goto END;        /* end all searching */

                   end for
               end for
           end for
       end for
 END:
```

Figure 3.2. Pseudocode describing the short hop procedure used in Tabu search.

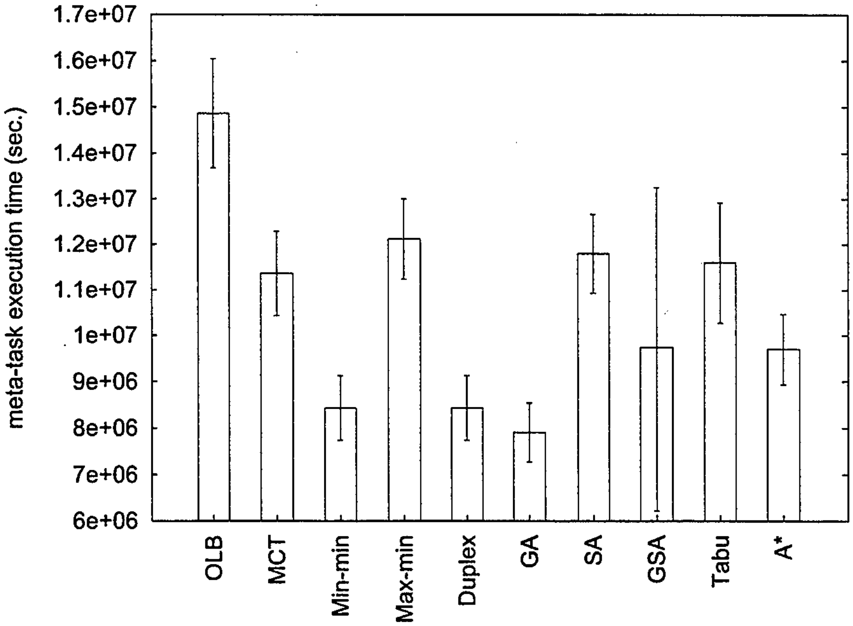


Figure 4.1. Consistent, high task, high machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean. For this case, the MET schedule was an order of magnitude worse than any other schedule and so is not shown.

	machines							
t	25,137.5	52,468.0	150,206.8	289,992.5	392,348.2	399,562.1	441,485.5	518,283.1
a	30,802.6	42,744.5	49,578.3	50,575.6	58,268.1	58,987.9	85,213.2	87,893.0
s	242,727.1	661,498.5	796,048.1	817,745.8	915,235.9	925,875.6	978,057.6	1,017,448.1
k	68,050.1	303,515.9	324,093.1	643,133.7	841,877.3	856,312.9	861,314.8	978,066.3
s	6,480.2	42,396.7	98,105.4	166,346.8	240,319.5	782,658.5	871,532.6	1,203,339.8
	175,953.8	210,341.9	261,825.0	306,034.2	393,292.2	412,085.4	483,691.9	515,645.9
	116,821.4	240,577.6	241,127.9	406,791.4	1,108,758.0	1,246,430.8	1,393,067.0	1,587,743.1
	36,760.6	111,631.5	150,926.0	221,390.0	259,491.1	383,709.7	442,605.7	520,276.8

Table 4.1. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with consistent, high task, high machine heterogeneity used in generating Figure 4.1.

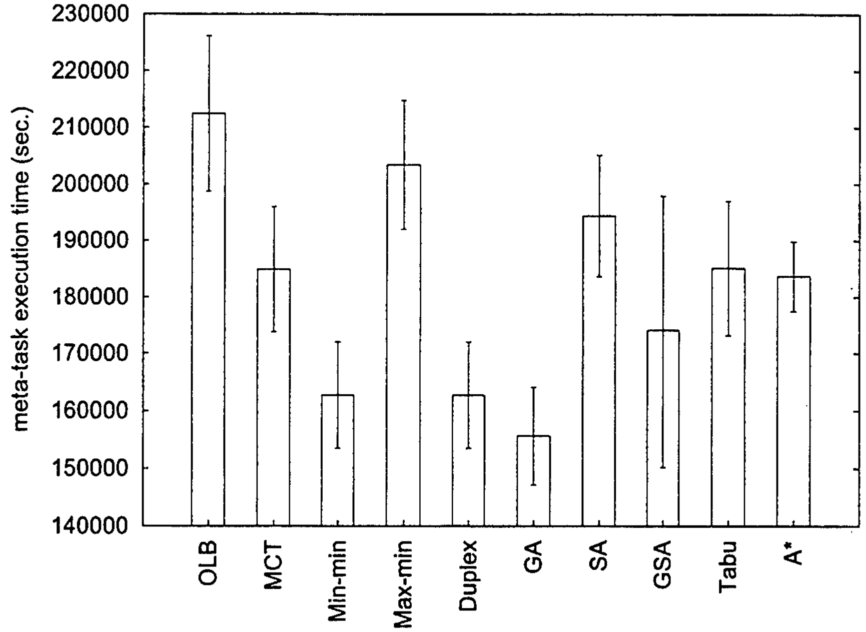


Figure 4.2. Consistent, high task, low machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean. For this case, the MET schedule was an order of magnitude worse than any other schedule and so is not shown.

	machines							
t	745.2	839.8	1,192.9	1,342.1	1,896.3	2,861.4	3,180.5	3,483.3
a	5,000.3	5,084.6	7,350.5	8,291.5	8,517.4	8,653.4	8,977.8	9,658.6
s	2,119.7	2,975.5	3,046.0	4,162.5	4,663.0	4,971.3	5,057.6	5,318.3
k	2,571.3	2,788.2	3,100.9	6,086.9	7,346.7	8,908.7	8,909.2	9,171.6
s	1,344.3	1,559.0	1,758.3	2,815.1	3,057.0	3,161.5	4,174.6	4,949.9
	4,479.1	6,283.3	8,735.4	9,241.4	12,022.0	12,079.3	14,165.8	15,684.7
	3,775.2	4,506.4	4,902.4	7,242.2	7,843.8	8,647.3	8,861.6	10,161.8
	2,227.6	5,199.6	5,896.1	6,316.3	10,079.8	10,175.9	10,630.7	10,977.6

Table 4.2. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with consistent, high task, low machine heterogeneity used in generating Figure 4.2.

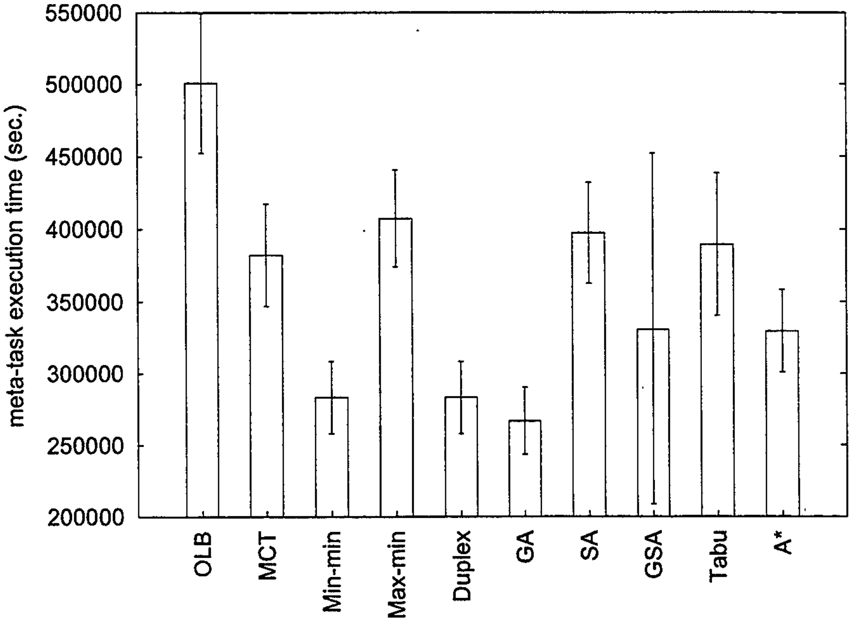


Figure 4.3. Consistent, low task, high machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean. For this case, the MET schedule was an order of magnitude worse than any other schedule and so is not shown.

	machines							
t	117.8	771.3	847.7	1,113.3	1,494.2	1,769.5	1,784.8	2,065.6
a	5,645.6	6,664.7	6,705.0	6,852.4	7,116.5	7,193.0	7,858.9	7,947.9
s	13,232.4	13,404.8	13,475.7	13,660.6	14,090.2	14,122.1	14,238.9	14,889.6
k	18,486.2	18,515.4	18,803.2	18,913.0	19,020.1	19,319.0	19,605.4	20,001.6
s	22,748.8	22,999.1	23,665.0	23,687.3	23,759.6	23,780.4	24,632.7	25,329.2
	28,511.5	29,095.5	30,172.9	30,239.7	30,695.7	30,854.2	30,886.1	31,261.5
	35,244.7	35,293.3	35,909.2	36,265.1	36,394.4	38,436.7	38,545.2	38,560.5
	41,086.6	41,133.9	41,359.1	41,798.4	41,893.0	42,235.0	42,641.0	42,692.4

Table 4.3. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with consistent, low task, high machine heterogeneity used in generating Figure 4.3.

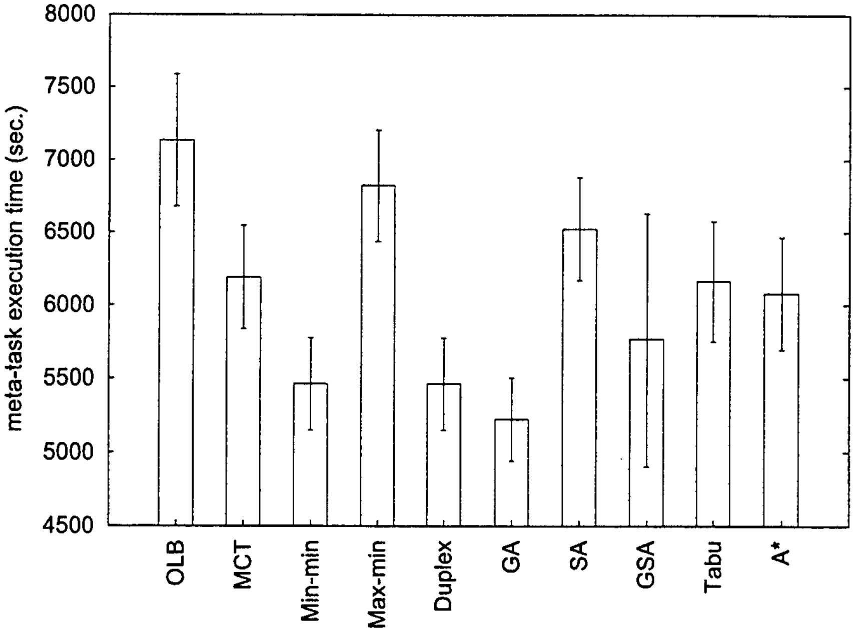


Figure 4.4. Consistent, low task, low machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean. For this case, the MET schedule was an order of magnitude worse than any other schedule and so is not shown.

	machines							
t	70.1	111.7	117.6	118.7	152.9	155.3	175.4	177.4
a	55.4	70.6	72.5	121.2	131.8	142.9	207.1	241.9
s	104.0	106.8	118.7	152.3	156.0	170.0	193.0	258.4
k	113.6	161.2	186.4	260.0	274.1	366.5	369.0	370.4
s	46.0	53.0	54.5	62.7	68.6	131.5	141.2	143.5
	29.5	33.2	80.5	108.8	110.8	119.4	133.0	152.3
	60.9	73.3	77.8	92.8	102.5	134.0	147.9	161.4
	75.2	111.9	204.2	270.3	293.9	304.4	408.7	429.1

Table 4.4. Sample 8×8 excerpt from one of the 512×16 *ETC* matrices with consistent, low task, low machine heterogeneity used in generating Figure 4.4.

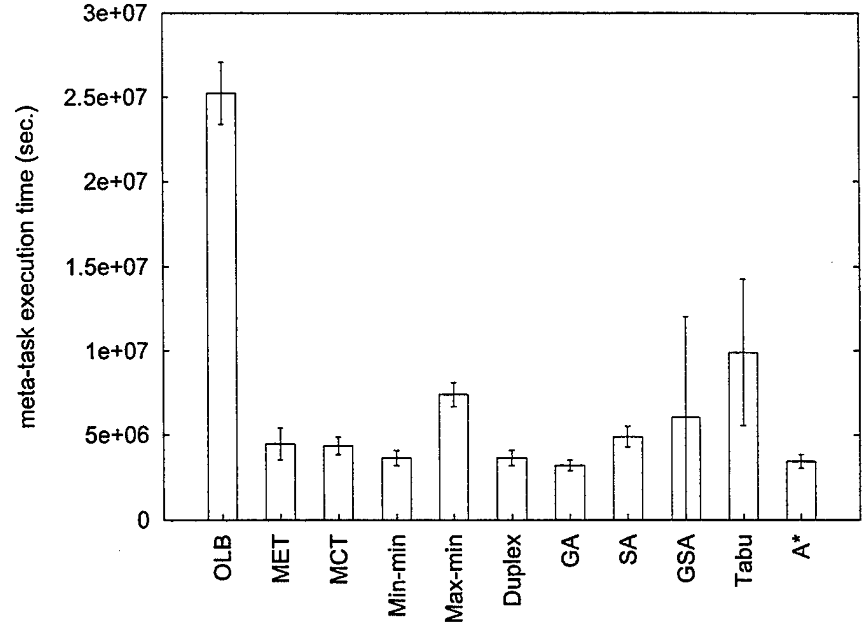


Figure 4.5. Inconsistent, high task, high machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	436,735.9	815,309.1	891,469.0	1,722,197.6	1,340,988.1	740,028.0	1,749,673.7	251,140.1
a	950,470.7	933,830.1	2,156,144.2	2,202,018.0	2,286,210.0	2,779,669.0	220,536.3	1,769,184.5
s	453,126.6	479,091.9	150,324.5	386,338.1	401,682.9	218,826.0	242,699.6	11,392.2
k	1,289,078.2	1,400,308.1	2,378,363.0	2,458,087.0	351,387.4	925,070.1	2,097,914.2	1,206,158.2
s	646,129.6	576,144.9	1,475,908.2	424,448.8	576,238.7	223,453.8	256,804.5	88,737.9
	1,061,682.3	43,439.8	1,355,855.5	1,736,937.1	1,624,942.6	2,070,705.1	1,977,650.2	1,066,470.8
	10,783.8	7,453.0	3,454.4	23,720.8	29,817.3	1,143.7	44,249.2	5,039.5
	1,940,704.5	1,682,338.5	1,978,545.6	788,342.1	1,192,052.5	1,022,914.1	701,336.3	1,052,728.3

Table 4.5. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with inconsistent, high task, high machine heterogeneity used in generating Figure 4.5.

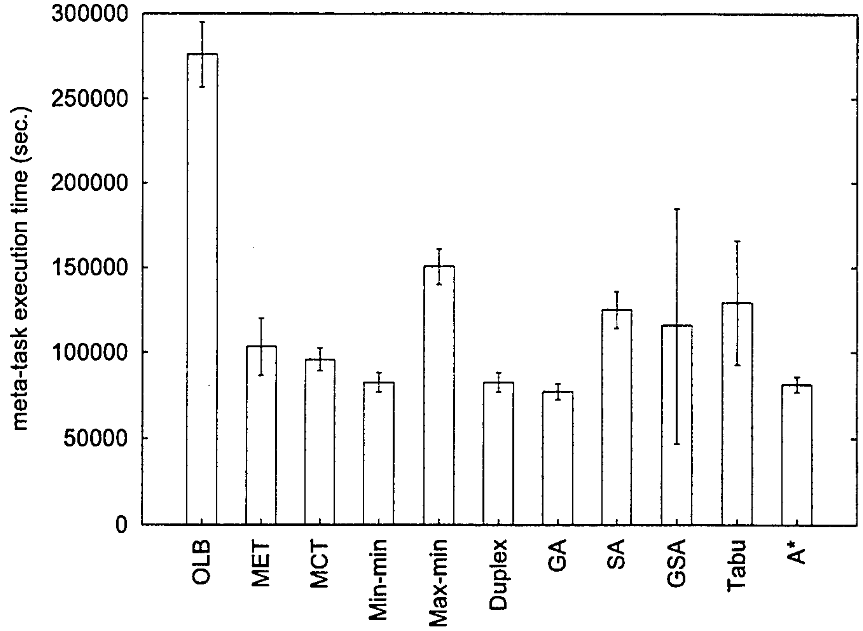


Figure 4.6. Inconsistent, high task, low machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	21,612.6	13,909.7	6,904.1	3,621.5	3,289.5	8,752.0	5,053.7	14,515.3
a	578.4	681.1	647.9	477.1	811.9	619.5	490.9	828.7
s	122.8	236.9	61.3	143.6	56.0	313.4	283.5	241.9
k	1,785.7	1,528.1	6,998.8	4,265.3	3,174.6	3,438.0	7,168.4	2,059.3
s	510.8	472.0	358.5	461.4	1,898.7	1,535.4	1,810.2	906.6
	22,916.7	18,510.0	11,932.7	6,088.3	9,239.7	15,036.4	18,107.7	12,262.6
	5,985.3	2,006.5	1,546.4	6,444.6	2,640.0	7,389.3	5,924.9	1,867.2
	16,192.4	3,088.9	16,532.5	13,160.6	10,574.2	7,136.3	15,353.4	2,150.6

Table 4.6. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with inconsistent, high task, low machine heterogeneity used in generating Figure 4.6.

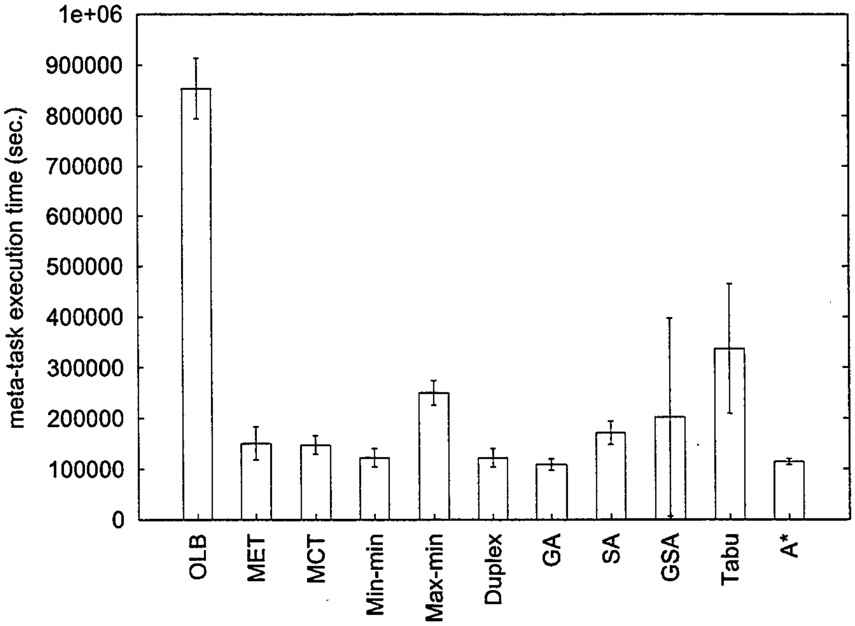


Figure 4.7. Inconsistent, low task, high machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	16,603.2	71,369.1	39,849.0	44,566.1	55,124.3	9,077.3	87,594.5	31,530.5
a	738.3	2,375.0	5,606.2	804.9	1,535.8	4,772.3	994.2	1,833.9
s	1,513.8	45.1	1,027.3	2,962.1	2,748.2	2,406.3	19.4	969.9
k	2,219.9	5,989.2	2,747.0	88.2	2,055.1	665.0	356.3	2,404.9
s	12,654.7	10,483.7	10,601.5	6,804.6	134.3	10,532.8	12,341.5	5,046.3
	4,226.0	48,152.2	11,279.3	35,471.1	30,723.4	24,234.0	6,366.9	22,926.9
	20,668.5	28,875.9	29,610.1	7,363.3	24,488.0	31,077.3	8,705.0	11,849.4
	52,953.2	14,608.1	58,137.2	16,685.5	36,571.3	35,888.8	38,147.0	15,167.5

Table 4.7. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with inconsistent, low task, high machine heterogeneity used in generating Figure 4.7.

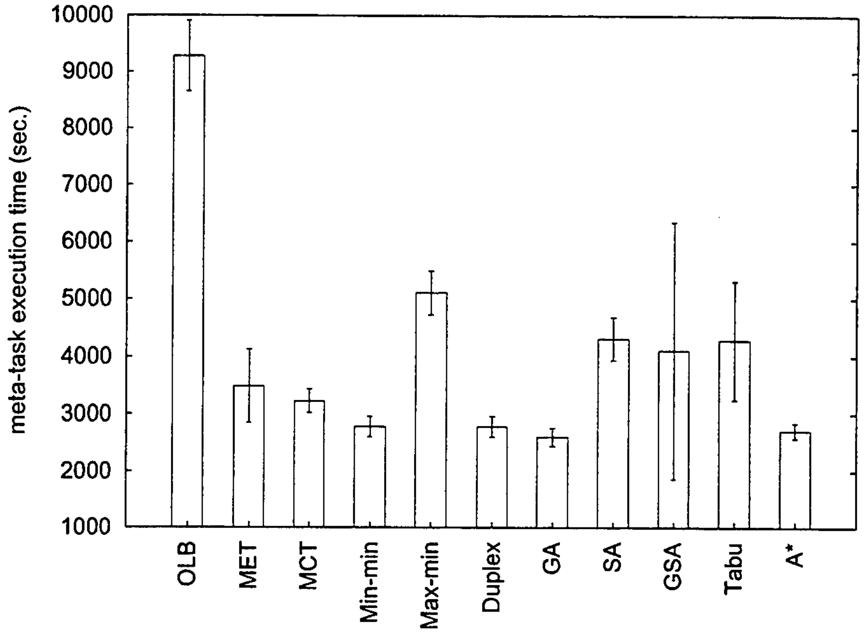


Figure 4.8. Inconsistent, low task, low machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	512.9	268.0	924.9	494.4	611.2	606.9	921.6	209.6
a	8.5	16.8	23.4	19.2	27.9	22.7	19.6	8.3
s	228.8	238.5	107.2	180.0	334.6	88.2	192.8	125.7
k	345.1	642.4	136.8	206.2	559.5	349.5	640.2	664.2
s	117.3	235.9	149.9	71.5	136.6	363.6	182.8	359.5
	240.7	412.0	259.1	319.8	237.5	338.3	178.5	537.7
	462.8	93.3	574.9	449.4	421.8	559.6	487.7	298.7
	119.5	36.7	224.2	194.2	176.5	156.8	182.7	192.0

Table 4.8. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with inconsistent, low task, low machine heterogeneity used in generating Figure 4.8.

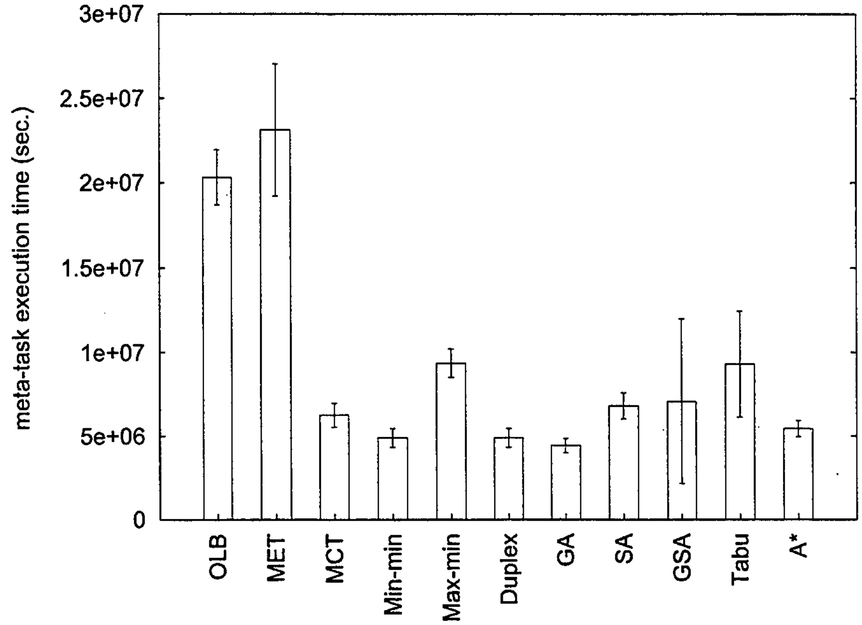


Figure 4.9. Partially-consistent, high task, high machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	1,003,569.7	910,811.9	1,085,529.8	1,646,242.8	1,087,655.5	2,121,084.5	1,141,898.7	749,952.3
a	27,826.6	409,936.4	168,341.7	858,511.3	353,691.8	270,449.8	420,799.6	152,786.0
s	8,415.4	101,202.5	16,453.7	64,152.5	29,172.8	36,738.5	61,114.5	142,411.2
k	17,050.5	195,067.8	79,175.8	787,263.3	173,239.2	438,599.0	378,563.4	747,305.4
s	32,275.4	434,445.7	135,989.1	496,326.8	221,097.9	463,577.7	244,747.3	431,704.5
	28,850.0	138,449.0	32,730.9	93,025.9	90,044.4	223,827.9	96,715.5	129,979.1
	145,038.5	350,917.4	210,957.4	265,590.5	486,217.7	317,915.2	728,732.4	625,365.5
	11,763.0	460,975.2	214,456.3	821,904.1	296,960.4	459,109.0	350,026.7	54,926.4

Table 4.9. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with partially-consistent, high task, high machine heterogeneity used in generating Figure 4.9.

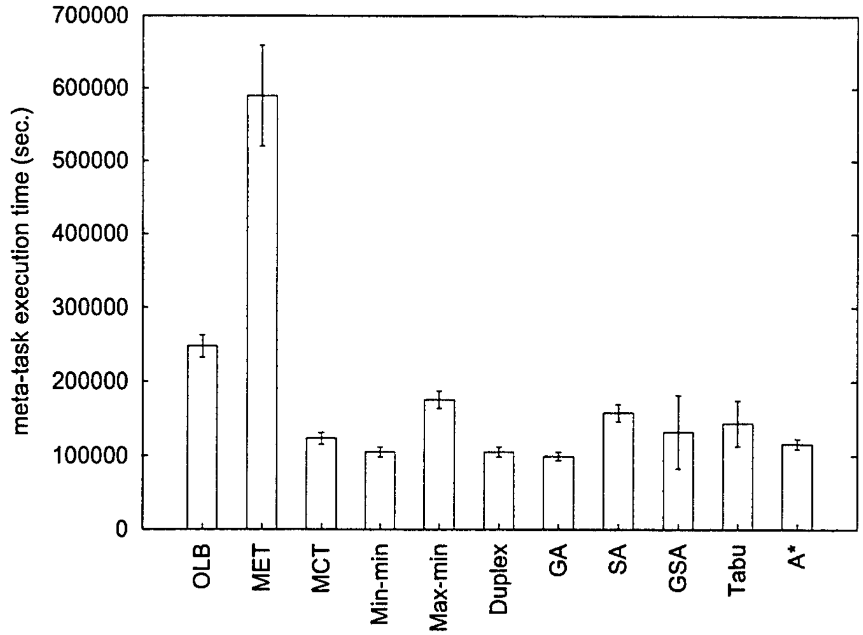


Figure 4.10. Partially-consistent, high task, low machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	2,312.2	3,186.4	2,475.5	10,455.3	3,749.3	11,879.5	4,594.3	1,861.9
a	3,403.7	16,572.1	6,503.7	5,764.5	12,108.2	19,655.2	13,769.3	16,726.1
s	5,909.0	17,499.1	9,042.4	25,581.2	11,604.0	9,846.1	12,502.8	12,182.2
k	1,911.0	10,251.3	3,551.2	11,450.1	4,710.2	5,633.8	4,900.0	7,485.6
s	2,303.6	5,952.0	2,468.3	7,128.6	2,616.6	7,028.0	4,622.8	8,640.4
	6,866.3	2,723.1	8,230.5	14,167.8	9,109.1	16,271.5	9,376.5	20,782.4
	3,968.7	3,954.7	7,130.2	10,055.4	11,557.9	13,028.4	14,230.1	3,955.8
	3,250.5	14,124.1	4,099.1	16,093.4	4,845.7	5,201.4	5,756.0	7,354.7

Table 4.10. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with partially-consistent, high task, low machine heterogeneity used in generating Figure 4.10.

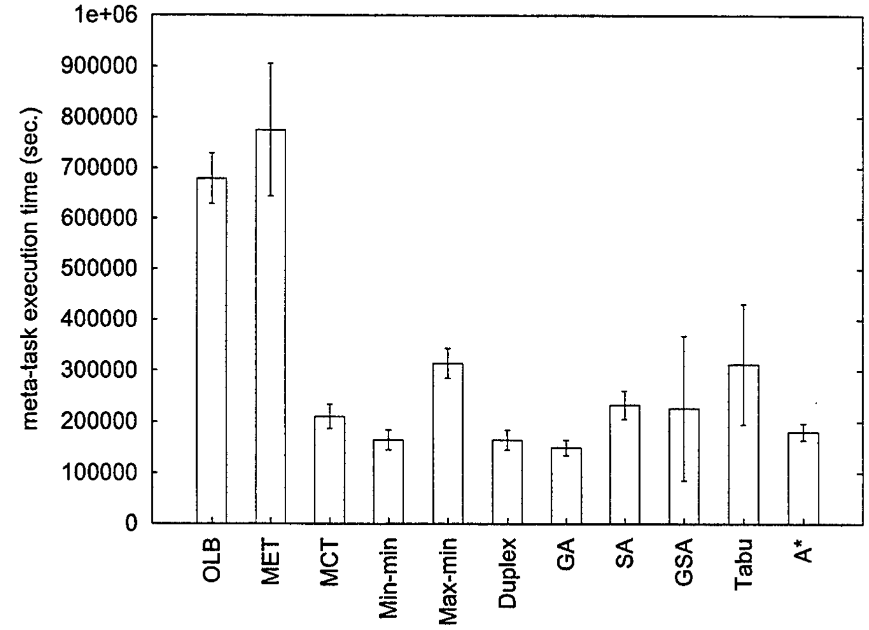


Figure 4.11. Partially-consistent, low task, high machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	173.9	1,262.8	438.4	174.5	539.4	216.9	701.3	931.2
a	3,007.7	14,169.0	3,075.9	3,810.9	13,178.0	30,292.9	18,849.8	18,687.7
s	1,187.5	9,948.8	4,700.4	17,941.7	7,057.8	4,495.1	8,449.5	8,212.0
k	2,342.0	2,938.6	5,212.7	11,842.0	5,946.4	5,816.1	7,481.9	3,923.8
s	82.2	9,957.8	8,950.4	57,354.7	9,369.5	10,626.8	10,286.4	52,394.2
	4,746.0	26,994.2	10,501.9	64,684.6	12,482.4	57,055.0	16,125.6	40,044.1
	464.9	1,363.6	508.7	1,692.6	913.7	3,953.8	1,159.5	3,660.2
	15,295.7	53,303.0	20,572.0	50,002.9	21,410.2	34,503.0	24,606.6	44,327.0

Table 4.11. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with partially-consistent, low task, high machine heterogeneity used in generating Figure 4.11.

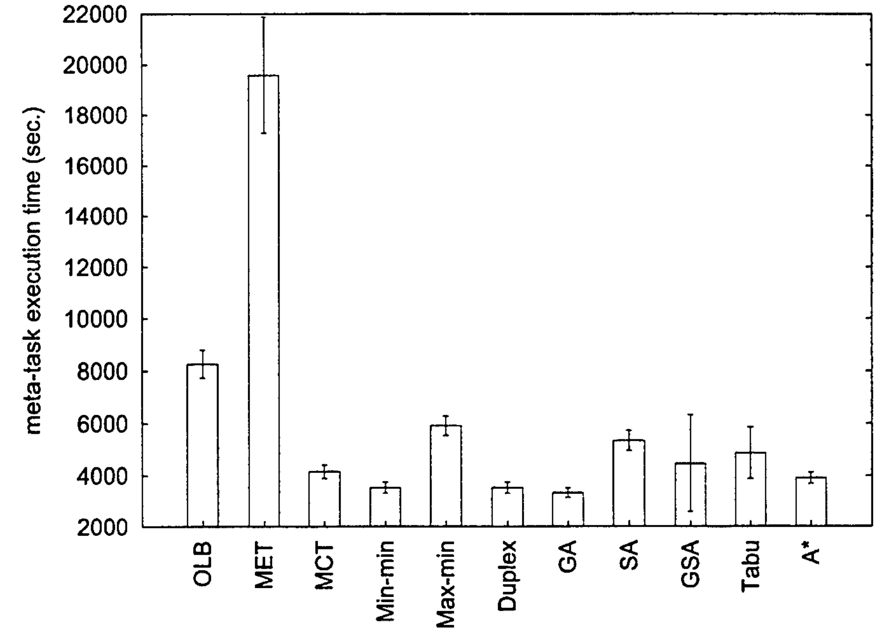


Figure 4.12. Partially-consistent, low task, low machine heterogeneity execution times for schedules from the eleven mapping heuristics, taken as the mean over 100 *ETC* matrices (trials). For each trial there are 512 tasks and 16 machines. For each heuristic, the range bars show the 95 percent confidence interval for the mean.

	machines							
t	90.5	703.0	148.2	736.7	151.0	251.2	177.4	593.6
a	47.5	329.2	65.5	61.0	121.6	91.5	144.9	72.8
s	107.8	544.4	179.5	309.4	247.1	287.7	380.9	143.2
k	62.0	203.2	69.2	61.7	92.4	55.7	93.8	221.3
s	159.7	823.7	160.1	560.7	392.9	133.7	603.9	621.3
	94.5	279.1	113.0	48.7	139.0	167.8	230.8	127.8
	93.9	175.2	413.7	144.6	489.2	612.9	541.9	755.4
	109.5	503.1	226.0	213.0	601.9	812.5	709.5	238.0

Table 4.12. Sample 8 × 8 excerpt from one of the 512 × 16 *ETC* matrices with partially-consistent, low task, low machine heterogeneity used in generating Figure 4.12.

A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems

Tracy D. Braun[†], Howard Jay Siegel[†], Noah Beck[†], Ladislau L. Bölöni[‡],
Muthucumaru Maheswaran[§], Albert I. Reuther[†], James P. Robertson[*], Mitchell D. Theys[†],
Bin Yao[†], Debra Hensgen[◇], and Richard F. Freund[¶]

[†]School of Electrical and Computer Engineering
[‡]Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
{tdbraun, hj, noah, reuther, theys, yaob}
@ecn.purdue.edu, boloni@cs.purdue.edu

[§]Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2 Canada
maheswar@cs.umanitoba.ca

[*]Motorola
6300 Bridgepoint Parkway
Bldg. #3, MD: OE71
Austin, TX 78730 USA
robertso@ibmoto.com

[◇]Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118 USA
hensgen@cs.nps.navy.mil

[¶]NOEMIX
1425 Russ Blvd., Ste. T-110
San Diego, CA 92101 USA
rffreund@noemix.com

Abstract

Heterogeneous computing (HC) environments are well suited to meet the computational demands of large, diverse groups of tasks (i.e., a meta-task). The problem of mapping (defined as matching and scheduling) these tasks onto the machines of an HC environment has been shown, in general, to be NP-complete, requiring the development of heuristic techniques. Selecting the best heuristic to use in a given environment, however, remains a difficult problem, because comparisons are often clouded by different underlying assumptions in the original studies of each heuristic. Therefore, a collection of eleven heuristics from the literature has been selected, implemented, and analyzed under one set of common assumptions. The eleven heuristics examined are Opportunistic Load Balancing, User-Directed Assignment, Fast Greedy, Min-min, Max-min, Greedy, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A. This study provides one even basis for comparison and insights into circumstances where one technique will outperform another. The evaluation procedure is specified, the heuristics are defined, and then selected results are compared.*

1. Introduction

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links to execute different computationally intensive applications that have diverse computational requirements [10, 18, 24]. The general problem of mapping (i.e., matching and scheduling) tasks to machines has been shown to be NP-complete [8, 15]. Heuristics developed to perform this mapping function are often difficult to compare because of different underlying assumptions in the original studies of each heuristic [3]. Therefore, a collection of eleven heuristics from the literature has been selected, implemented, and compared by simulation studies under one set of common assumptions.

To facilitate these comparisons, certain simplifying assumptions were made. Let a meta-task be defined as a collection of independent tasks with no data dependencies (a given task, however, may have subtasks and dependencies among the subtasks). For this case study, it is assumed that static (i.e., off-line or predictive) mapping of meta-tasks is being performed. (In some systems, all tasks and subtasks in a meta-task, as defined above, are referred to as just tasks.)

It is also assumed that each machine executes a single task at a time, in the order in which the tasks ar-

This research was supported in part by the DARPA/ITO Quorum Program under NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448. Some of the equipment used was donated by Intel.

rived. Because there are no dependencies among the tasks, scheduling is simplified, and thus the resulting solutions of the mapping heuristics focus more on finding an efficient matching of tasks to machines. It is also assumed that the size of the meta-task (number of tasks to execute), $\underline{t}$, and the number of machines in the HC environment, $\underline{m}$, are static and known *a priori*.

Section 2 defines the computational environment parameters that were varied in the simulations. Descriptions of the eleven mapping heuristics are found in Section 3. Section 4 examines selected results from the simulation study. A list of implementation parameters and procedures that could be varied for each heuristic is presented in Section 5.

This research was supported in part by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks) [13]. MSHN is a collaborative research effort among the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver requested qualities of service. The heuristics developed in this paper or their derivatives may be included in the Scheduling Advisor component of the MSHN prototype.

2. Simulation Model

The eleven static mapping heuristics were evaluated using simulated execution times for an HC environment. Because these are static heuristics, it is assumed that an accurate estimate of the expected execution time for each task on each machine is known prior to execution and contained within an *ETC* (expected time to compute) matrix. One row of the *ETC* matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the *ETC* matrix consists of the estimated execution times of a given machine for each task in the meta-task. Thus, $ETC(i, j)$ is the estimated execution time for task i on machine j. (These times are assumed to include the time to move the executables and data associated with each task to the particular machine when necessary.) The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems (e.g., [11, 16, 25]). (Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [14, 24].)

For the simulation studies, characteristics of the *ETC* matrices were varied in an attempt to represent a variety of possible HC environments. The *ETC* matrices used were generated using the following method. Initially, a $t \times 1$ baseline column vector, $\underline{B}$, of floating point values is created. Let ϕ_b be the upper-bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $x_b^i \in [1,\ \phi_b)$, and letting $B(i) = x_b^i$ for $0 \leq i < t$. Next, the rows of the *ETC* matrix are constructed. Each element $ETC(i, j)$ in row i of the *ETC* matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper-bound of ϕ_r. This new random number, $x_r^{i,j} \in [1,\ \phi_r)$, is called a row multiplier. One row requires m different row multipliers, $0 \leq j < m$. Each row i of the *ETC* matrix can be then described as $ETC(i, j) = B(i) \times x_r^{i,j}$, for $0 \leq j < m$. (The baseline column itself does not appear in the final *ETC* matrix.) This process is repeated for each row until the $m \times t$ *ETC* matrix is full. Therefore, any given value in the *ETC* matrix is within the range $[1,\ \phi_b \times \phi_r)$.

To evaluate the heuristics for different mapping scenarios, the characteristics of the *ETC* matrix were varied based on several different methods from [2]. The amount of variance among the execution times of tasks in the meta-task for a given machine is defined as task heterogeneity. Task heterogeneity was varied by changing the upper-bound of the random numbers within the baseline column vector. High task heterogeneity was represented by $\phi_b = 3000$ and low task heterogeneity used $\phi_b = 100$. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the upper-bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using $\phi_r = 1000$, while low machine heterogeneity values used $\phi_r = 10$. These heterogeneous ranges are based on one type of expected environment for MSHN.

To further vary the *ETC* matrix in an attempt to capture more aspects of realistic mapping situations, different *ETC* matrix consistences were used. An *ETC* matrix is said to be consistent if whenever a machine j executes any task i faster than machine k, then machine j executes all tasks faster than machine k [2]. Consistent matrices were generated by sorting each row of the *ETC* matrix independently. In contrast, inconsistent matrices characterize the situation where machine j is faster than machine k for some tasks, and slower for others. These matrices are left in the unordered, random state in which they were generated. In between these two extremes, semi-consistent matrices represent a partial ordering among the machine/task execution times. For the semi-consistent matrices used here, the row elements in column positions $\{0, 2, 4, \ldots\}$ of row i are extracted, sorted, and replaced in order, while the row elements in column positions $\{1, 3, 5, \ldots\}$ remain unordered. (That is, the even columns are consistent and the odd columns are inconsistent.)

Sample *ETC* matrices for the four inconsistent heterogeneous permutations of the characteristics listed above are shown in Tables 1 through 4. (Other probability distributions for *ETC* values, including an exponential distribution and a truncated Gaussian [1] distribution, were also investigated, but not included in the results discussed here.) All results in this study used *ETC* matrices that were of size $t = 512$ tasks by $m = 16$ machines. While it was necessary to select some specific parameter values to allow implementation of a simulation, the characteristics and techniques presented here are completely general. Therefore, if these parameter values do not apply to a specific situation of interest, researchers may substitute in their own ranges, distributions, matrix sizes, etc., and the evaluation software of this study will still apply.

3. Heuristic Descriptions

The definitions of the eleven static meta-task mapping heuristics are provided below. First, some preliminary terms must be defined. Machine availability time, $mat(j)$, is the earliest time a machine j can complete the execution of all the tasks that have previously been assigned to it. Completion time, $ct(i, j)$, is the machine availability time plus the execution time of task i on machine j, i.e., $ct(i, j) = mat(j) + ETC(i, j)$. The performance criterion used to compare the results of the heuristics is the maximum value of $ct(i, j)$, for $0 \leq i < t$ and $0 \leq j < m$, for each mapping, also known as the makespan [19]. Each heuristic is attempting to minimize the makespan (i.e., finish execution of the meta-task as soon as possible).

The descriptions below implicitly assume that the machine availability times are updated after each task is mapped. For cases when tasks can be considered in an arbitrary order, the order in which the tasks appeared in the *ETC* matrix was used. Some of the heuristics listed below had to be modified from their original implementation to better handle the scenarios under consideration.

For many of the heuristics, there are control parameter values and/or control function specifications that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and/or information in the literature. These parameters and functions are mentioned in Section 5.

OLB: Opportunistic Load Balancing (OLB) assigns each task, in arbitrary order, to the next available machine, regardless of the task's expected execution time on that machine [1, 9, 10].

UDA: In contrast to OLB, User-Directed Assignment (UDA) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine's availability [1]. UDA is sometimes referred to as Limited Best Assignment (LBA), as in [1, 9].

Fast Greedy: Fast Greedy assigns each task, in arbitrary order, to the machine with the minimum completion time for that task [1].

Min-min: The Min-min heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion times, $M = \{m_i : m_i = \min_{0 \leq j < m}(ct(i, j))$, for each $i \in U\}$, is found. Next, the task with the overall *minimum* completion time from M is selected and assigned to the corresponding machine (hence the name Min-min). Lastly, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., $U = \emptyset$) [1, 9, 15].

Intuitively, Min-min attempts to map as many tasks as possible to their first choice of machine (on the basis of completion time), under the assumption that this will result in a shorter makespan. Because tasks with shorter execution times are being mapped first, it was expected that the percentage of tasks that receive their first choice of machine would generally be higher for Min-min than for Max-min (defined next), and this was verified by data recorded during the simulations.

Max-min: The Max-min heuristic is very similar to Min-min. The Max-min heuristic also begins with the set U of all unmapped tasks. Then, the set of minimum completion times, $M = \{m_i : m_i = \min_{0 \leq j < m}(ct(i, j))$, for each $i \in U\}$, is found. Next, the task with the overall *maximum* completion time from M is selected and assigned to the corresponding machine (hence the name Max-min). Lastly, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., $U = \emptyset$) [1, 9, 15].

The motivation behind Max-min is to attempt to minimize the penalties incurred by delaying the scheduling of long-running tasks. Assume that the meta-task being mapped has several tasks with short execution times, and a small quantity of tasks with very long execution times. Mapping the tasks with the longer execution times to their best machines first allows these tasks to be executed concurrently with the remaining tasks (with shorter execution times). This concurrent execution of long and short tasks can be more beneficial than a Min-min mapping where all of the shorter tasks would execute first, and then a few longer running tasks execute while several machines sit idle. The assumption here is that with Max-min the tasks with shorter execution times can be mixed with longer tasks and evenly distributed among the machines, resulting in better machine utilization and a better meta-task makespan.

Greedy: The Greedy heuristic is literally a combination of the Min-min and Max-min heuristics. The Greedy heuristic performs both of the Min-min and Max-min heuristics, and uses the better solution [1, 9].

GA: Genetic Algorithms (GAs) are a popular technique used for searching large solution spaces (e.g., [25, 27]). The version of the heuristic used for this study was adapted from [27] for this particular solution space. Figure 1 shows the steps in a general Genetic Algorithm.

The Genetic Algorithm implemented here operates on a population of 200 chromosomes (possible mappings) for a given meta-task. Each chromosome is a $t \times 1$ vector, where position i $(0 \leq i < t)$ represents task i, and the entry in position i is the machine to which the task has been mapped. The initial population is generated using two methods: (a) 200 randomly generated chromosomes from a uniform distribution, or (b) one chromosome that is the Min-min solution and 199 random solutions (mappings). The latter method is called seeding the population with a Min-min chromosome. The GA actually executes eight times (four times with initial populations from each method), and the best of the eight mappings is used as the final solution.

After the generation of the initial population, all of the chromosomes in the population are evaluated (i.e., ranked) based on their fitness value (i.e., makespan), with a smaller fitness value being a better mapping. Then, the main loop in Figure 1 is entered and a rank-based roulette wheel scheme [26] is used for selection. This scheme probabilistically generates new populations, where better mappings have a higher probability of surviving to the next generation. Elitism, the property of guaranteeing the best solution remains in the population [20], was also implemented.

Next, the crossover operation selects a pair of chromosomes and chooses a random point in the first chromosome. For the sections of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding tasks. Every chromosome is considered for crossover with a probability of 60%.

After crossover, the mutation operation is performed. Mutation randomly selects a task within the chromosome, and randomly reassigns it to a new machine. Both of these random operations select values from a uniform distribution. Every chromosome is considered for mutation with a probability of 40%.

Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when any one of three conditions are met: (a) 1000 total iterations, (b) no change in the elite chromosome for 150 iterations, or (c) all chromosomes converge. If no stopping criteria is met, the loop repeats, beginning with the selection of a new population. The stopping criteria that usually occurred in testing was no change in the elite chromosome in 150 iterations.

SA: Simulated Annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This solution uses the same representation for a solution as the chromosome for the GA.

SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [6, 17, 21]). This probability is based on a system temperature that decreases for each iteration. As the system temperature "cools," it is more difficult for currently poorer solutions to be accepted. The initial system temperature is the makespan of the initial mapping.

The specific SA procedure implemented here is as follows. The initial mapping is generated from a uniform random distribution. The mapping is mutated in the same manner as the GA, and the new makespan is evaluated. The decision algorithm for accepting or rejecting the new mapping is based on [6]. If the new makespan is better, the new mapping replaces the old one. If the new makespan is worse (larger), a uniform random number $\underline{z} \in [0,\ 1)$ is selected. Then, z is compared with $\underline{y}$, where

$$y = \frac{1}{1 + e^{\left(\frac{\text{old makespan–new makespan}}{\text{temperature}}\right)}}. \tag{1}$$

If $z > y$ the new (poorer) mapping is accepted, otherwise it is rejected, and the old mapping is kept.

Notice that for solutions with similar makespans (or if the system temperature is very large), $y \rightarrow 0.5$, and poorer solutions are more easily accepted. In contrast, for solutions with very different makespans (or if the system temperature is very small), $y \rightarrow 1$, and poorer solutions will usually be rejected.

After each mutation, the system temperature is decreased by 10%. This defines one iteration of SA. The heuristic stops when there is no change in the makespan for 150 iterations or the system temperature reaches zero. Most tests ended with no change in the makespan for 150 iterations.

GSA: The Genetic Simulated Annealing (GSA) heuristic is a combination of the GA and SA techniques [4, 23]. In general, GSA follows procedures similar to the GA outlined above. GSA operates on a population of 200 chromosomes, uses a Min-min seed in four out of eight initial populations, and performs similar mutation and crossover operations. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simplified SA decision process for accepting or rejecting a new chromosomes. GSA also used elitism to guarantee that the best solution always remained in the population.

The initial system temperature for the GSA selection process was set to the average makespan of the initial population, and decreased 10% for each iteration. When a new (post-mutation, post-crossover, or both) chromosome is compared with the corresponding original chromosome, if the new makespan is less than the old makespan plus the system temperature, then the new chromosome is accepted. That is, if

$$\text{new makespan} < (\text{old makespan} + \text{temperature}) \quad (2)$$

is true, the new chromosome becomes part of the population. Otherwise, the original chromosome survives to the next iteration. Therefore, as the system temperature decreases, it is again more difficult for poorer solutions (i.e., longer makespans) to be accepted. The two stopping criteria used were either (a) no change in the elite chromosome in 150 iterations or (b) 1000 total iterations. Again, the most common stopping criteria was no change in the elite chromosome in 150 iterations.

Tabu: Tabu search is a solution space search that keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas [7, 12]. A solution (mapping) uses the same representation as a chromosome in the GA approach.

The implementation of Tabu search used here begins with a random mapping, generated from a uniform distribution. Starting with the first task in the mapping, task $i = 0$, each possible pair of tasks is formed, $(i,\ j)$ for $0 \leq i < t-1$ and $i < j < t$. As each pair of tasks is formed, they exchange machine assignments. This constitutes a short hop. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. After each exchange, the new makespan is evaluated. If the new makespan is an improvement, the new mapping is accepted (a successful short hop), and the pair generation-and-exchange sequence starts over from the beginning ($i = 0$) of the new mapping. Otherwise, the pair generation-and-exchange sequence continues from its previous state, $(i,\ j)$. New short hops are generated until 1200 successful short hops have been made or all combinations of task pairs have been exhausted with no further improvement.

At this point, the final mapping from the local solution space search is added to the tabu list. The tabu list is a method of keeping track of the regions of the solution space that have already been searched. Next, a new random mapping is generated, and it must differ from each mapping in the tabu list by at least half of the machine assignments (a successful long hop). The intuitive purpose of a long hop is to move to a new region of the solution space that has not already been searched. The final stopping criterion for the heuristic is a total of 1200 successful hops (short and long combined). Then, the best mapping from the tabu list is the final answer.

A*: The final heuristic in the comparison study is known as the A* heuristic. A* has been applied to many other task allocation problems (e.g., [5, 16, 21, 22]). The technique used here is similar to [5].

A* is a tree search beginning at a root node that is usually a null solution. As the tree grows, intermediate nodes represent partial solutions (a subset of tasks are assigned to machines), and leaf nodes represent final solutions (all tasks are assigned to machines). The partial solution of a child node has one more task mapped than the parent node. Call this additional task $\underline{a}$. Each parent node generates m children, one for each possible mapping of a. After a parent node has done this, the parent node is removed and replaced in the tree by the m children. Based on experimentation and a desire to keep execution time of the heuristic tractable, the maximum number of nodes in the tree at any one time is limited in this study to $\underline{n_{max}} = 1024$.

Each node, $\underline{n}$, has a cost function, $\underline{f(n)}$, associated with it. The cost function is an estimated lower-bound on the makespan of the best solution that includes the partial solution represented by node n. Let $\underline{g(n)}$ represent the makespan of the task/machine assignments in the partial solution of node n, i.e., $g(n)$ is the maximum of the machine availability times ($mat(j)$) based on the set of tasks that have been mapped to machines in node n's partial solution. Let $\underline{h(n)}$ be a lower-bound estimate on the difference between the makespan of node n's partial solution and the makespan for the best complete solution that includes node n's partial solution. Then, the cost function for node n is computed as

$$f(n) = g(n) + h(n). \quad (3)$$

Therefore, $f(n)$ represents the makespan of the partial solution of node n plus a lower-bound estimate of the time to execute the rest of the (unmapped) tasks in the meta-task.

The function $h(n)$ is defined in terms of two functions, $h_1(n)$ and $h_2(n)$, which are two different approaches to deriving a lower-bound estimate. Recall that $M = \{m_i : m_i = \min_{0 \leq j < m}(ct(i,j))$, for each $i \in U\}$. For node n let $mmct(n)$ be the overall maximum element of M over all $i \in U$ (i.e., "the maximum minimum completion time"). Intuitively, $mmct(n)$ represents the best possible meta-task makespan by making the typically unrealistic assumption that each task in U can be assigned to the machine indicated in M without conflict. Thus, based on [5], $h_1(n)$ is defined as

$$h_1(n) = \max(0,\ (mmct(n) - g(n))). \qquad (4)$$

Next, let $sdma(n)$ be the sum of the differences between $g(n)$ and each machine availability time over all machines after executing all of the tasks in the partial solution represented by node n:

$$sdma(n) = \sum_{j=0}^{m-1}(g(n) - mat(j)). \qquad (5)$$

Intuitively, $sdma(n)$ represents the amount of machine availability time remaining that can be scheduled without increasing the final makespan. Let $smet(n)$ be defined as the sum of the minimum expected execution times (i.e., ETC values) for all tasks in U:

$$smet(n) = \sum_{i \in U}(\min_{0 \leq j < m}(ETC(i,j)) \qquad (6)$$

This gives an estimate of the amount of remaining work to do, which could increase the final makespan. The function h_2 is then defined as

$$h_2(n) = \max(0,\ (smet(n) - sdma(n))/m), \qquad (7)$$

where $(smet(n) - sdma(n))/m$ represents an estimate of the minimum increase in the meta-task makespan if the tasks in U could be "ideally" (but, in general, unrealistically) distributed among the machines. Using these definitions,

$$h(n) = \max(h_1(n), h_2(n)), \qquad (8)$$

representing a lower-bound estimate on the time to execute the tasks in U.

Thus, beginning with the root, the node with the minimum $f(n)$ is replaced by its m children, until n_{max} nodes are created. From that point on, any time a node is added, the tree is pruned by deleting the node with the largest $f(n)$. This process continues until a leaf node (representing a complete mapping) is reached. Note that if the tree is not pruned, this method is equivalent to an exhaustive search.

These eleven heuristics were all implemented under the common simulation model described in Section 2. The results from experiments using these implementations are described in the next section. Suggestions for alternative heuristic implementations are given in Section 5.

4. Experimental Results

An interactive software application has been developed that allows simulation, testing, and demonstration of the heuristics examined in Section 3 applied to the meta-tasks defined by the ETC matrices described in Section 2. The software allows a user to specify t and m, to select which ETC matrices to use, and to choose which heuristics to execute. It then generates the specified ETC matrices, executes the desired heuristics, and displays the results, similar to Figures 2 through 13. The results discussed in this section were generated using portions of this software.

When comparing mapping heuristics, the execution time of the heuristics themselves is an important consideration. For the heuristics listed, the execution times varied greatly. The experimental results discussed below were obtained on a Pentium II 400 MHz processor with 1GB of RAM. Each of the simpler heuristics (OLB, UDA, Fast Greedy, and Greedy) executed in a few seconds for one ETC matrix with $t = 512$ and $m = 16$. For the same sized ETC matrix, SA and Tabu, both of which manipulate a single solution during an iteration, averaged less than 30 seconds. GA and GSA required approximately 60 seconds per matrix because they manipulate entire populations, and A* required about 20 minutes per matrix.

The resulting meta-task execution times (makespans) from the simulations for every case of consistency, task heterogeneity, and machine heterogeneity are shown in Figures 2 through 13. All experimental results represent the execution time of a meta-task (defined by a particular ETC matrix) based on the mapping found by the heuristic specified, averaged over 100 different ETC matrices of the same type (i.e., 100 mappings). For each heuristic, the range bars show the minimum and maximum meta-task execution times over the 100 mappings (100 ETC matrices) used to compute the average meta-task execution time. Tables 1 through 4 show sample subsections from the four types of inconsistent ETC matrices considered. Semi-consistent and consistent matrices of the same types could be generated from these matrices as described in Section 2. For the results described here, however, entirely new matrices were generated for each case.

For the four consistent cases, Figures 2 through 5, the UDA algorithm had the worst execution times by an order of magnitude. This is easy to explain. For the consistent cases, all tasks will have the lowest execution time on one machine, and all tasks will be mapped to this particular machine. This corresponds to results found in [1]. Because of this poor performance, the UDA results were not included in Figures 2 through 5. OLB, Max-min, and SA had the next poorest results. GA performed the best for the consistent cases. This was due in part to the good performance of the Min-min heuristic. The best GA solution always came from one of the populations that had been seeded with the Min-min solution. As is apparent in the figures, Min-min performed very well on its own, giving the second best results. The mutation, crossover, and selection operations of the GA were always able to improve on this solution, however. GSA, which also used a Min-min seed, did not always improve upon the Min-min solution. Because of the probabilistic procedure used during selection, GSA would sometimes accept poorer intermediate solutions. These poorer intermediate solutions never led to better final solutions, thus GSA gave the third best results. The performance of A* was hindered because the estimates made by $h_1(n)$ and $h_2(n)$ are not as accurate for consistent cases as they are for inconsistent and semi-consistent cases. For consistent cases, $h_1(n)$ underestimates the competition for machines and $h_2(n)$ underestimates the "workload" distributed to each machine.

These results suggest that if the best overall solution is desired, the GA should be employed. However, the improvement of the GA solution over the Min-min solution was never more than 10%. Therefore, the Min-min hueristic may be more appropriate in certain situations, given the difference in execution times of the two heuristics.

For the four inconsistent test cases in Figures 6 through 9, UDA performs much better and the performance of OLB degrades. Because there is no pattern to the consistency, OLB will assign more tasks to poor or even worst-case machines, resulting in poorer schedules. In contrast, UDA improves because the "best" machines are distributed across the set of machines, thus task assignments will be more evenly distributed among the set of machines avoiding load imbalance. Similarly, Fast Greedy and Min-min performed very well, and slightly outperformed UDA, because the machines providing the best task completion times are more evenly distributed among the set of machines. Min-min was also better than Max-min for all of the inconsistent cases. The advantages Min-min gains by mapping "best case" tasks first outweighs the savings in penalties Max-min has by mapping "worst case" tasks first.

Tabu gave the second poorest results for the inconsistent cases, at least 16% poorer than the other heuristics. Inconsistent matrices generated more successful short hops than the associated consistent matrices. Therefore, fewer long hops were generated and less of the solution space was searched, resulting in poorer solutions. The increased number of successful short hops for inconsistent matrices can be explained as follows. The pairwise comparison procedure used by the short hop procedure will assign machines with better performance first, early in the search procedure. For the consistent cases, these machines will always be from the same set of machines. For inconsistent cases, these machines could be any machine. Thus, for consistent cases, the search becomes somewhat ordered, and the successful short hops get exhausted faster. For inconsistent cases, the lack of order means there are more successful short hops, resulting in fewer long hops.

GA and A* had the best average makespans, and were usually within a small constant factor of each other. The random approach employed by these methods was useful and helped overcome the difficulty of locating good mappings within inconsistent matrices. GA again benefited from having the Min-min initial mapping. A* did well because if the tasks get more evenly distributed among the machines, this more closely matches the lower-bound estimates of $h_1(n)$ and $h_2(n)$.

Finally, consider the semi-consistent cases in Figures 10 through 13. For semi-consistent cases with high machine heterogeneity, the UDA heuristic again gave the worst results. Intuitively, UDA is suffering from the same problem as in the consistent cases: half of all tasks are getting assigned to the same machine. OLB does poorly for high machine heterogeneity cases because worst case matchings will have higher execution times for high machine heterogeneity. For low machine heterogeneity, the worst case matchings have a much lower penalty. The best heuristics for the semi-consistent cases were Min-min and GA. This is not surprising because these were two of the best heuristics from the consistent and inconsistent tests, and semi-consistent matrices are a combination of consistent and inconsistent matrices. Min-min was able to do well because it searched the entire row for each task and assigned a high percentage of tasks to their first choice. GA was robust enough to handle the consistent components of the matrices, and did well for the same reasons mentioned for inconsistent matrices.

5. Alternative Implementations

The experimental results in Section 4 show the performance of each heuristic under the assumptions presented. For several heuristics, specific control parameter values and control functions had to be selected. In most cases, control parameter values and control functions were based on the references cited or experiments conducted. However, for these heuristics, different, valid implementations are possible using different control parameters and control functions.

GA, SA, GSA: Several parameter values could be varied among these techniques, including (where appropriate) population size, crossover probability, mutation probability, stopping criteria, number of runs with different initial populations per result, and the system temperature. The specific procedures used for the following actions could also be modified (where appropriate) including initial population "seed" generation, mutation, crossover, selection, elitism, and the accept/reject new mapping procedure.

Tabu: The short hop method implemented was a "first descent" (take the first improvement possible) method. "Steepest descent" methods (where several short hops are considered simultaneously, and the one with the most improvement is selected) are also used in practice [7]. Other techniques that could be varied are the long hop method, the order of the short hop pair generation-and-exchange sequence, and the stopping condition. Two possible alternative stopping criteria are when the tabu list reaches a specified number of entries, or when there is no change in the best solution in a specified number of hops.

A*: Several variations of the A* method that was employed here could be implemented. Different functions could be used to estimate the lower-bound $h(n)$. The maximum size of the search tree could be varied, and several other techniques exist for tree pruning (e.g., [21]).

In summary, for the GA, SA, GSA, Tabu, and A* heuristics there are a great number of possible valid implementations. An attempt was made to use a reasonable implementation of each heuristic for this study. Future work could examine other implementations.

6. Conclusions

The goal of this study was to provide a basis for comparison and insights into circumstances where one technique will out perform another for eleven different heuristics. The characteristics of the *ETC* matrices used as input for the heuristics and the methods used to generate them were specified. The implementation of a collection of eleven heuristics from the literature was described. The results of the mapping heuristics were discussed, revealing the best heuristics to use in certain scenarios. For the situations, implementations, and parameter values used here, GA was the best heuristic for most cases, followed closely by Min-min, with A* also doing well for inconsistent matrices.

A software tool was developed that allows others to compare these heuristics for many different types of *ETC* matrices. These heuristics could also be the basis of a mapping toolkit. If this toolkit were given an *ETC* matrix representing an actual meta-task and an actual HC environment, the toolkit could analyze the *ETC* matrix, and utilize the best mapping heuristic for that scenario. Depending on the overall situation, the execution time of the mapping heuristic itself may impact this decision. For example, if the best mapping available in less than one minute was desired and if the characteristics of a given *ETC* matrix most closely matched a consistent matrix, Min-min would be used; if more time was available for finding the best mapping, GA and A* should be considered.

The comparisons of the eleven heuristics and twelve situations provided in this study can be used by researchers as a starting point when choosing heuristics to apply in different scenarios. They can also be used by researchers for selecting heuristics to compare new, developing techniques against.

Acknowledgments – The authors thank Shoukat Ali and Taylor Kidd for their comments.

References

[1] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79–87.

[2] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Sept. 1997 (D. Hensgen, advisor.)

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1998, pp. 330–335

(included in the Proceedings of the *7th IEEE Symposium on Reliable Distributed Systems*, 1998).

[4] H. Chen, N. S. Flann, and D. W. Watson, "Parallel genetic simulated annealing: A massively parallel SIMD approach," *IEEE Transactions on Parallel and Distributed Computing*, Vol. 9, No. 2, Feb. 1998, pp. 126–136.

[5] K. Chow and B. Liu, "On mapping signal processing algorithms to a heterogeneous multiprocessor system," *1991 International Conference on Acoustics, Speech, and Signal Processing - ICASSP 91*, Vol. 3, May 1991, pp. 1585–1588.

[6] M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, Vol. 42, No. 6-7, Dec. 1996, pp. 465–475.

[7] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro, "Improving search by incorporating evolution principles in parallel tabu search," *1994 IEEE Conference on Evolutionary Computation*, Vol. 2, 1994, pp. 823–828.

[8] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[9] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184–199.

[10] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13–17.

[11] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[12] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, MA, June 1997.

[13] D. A. Hensgen, T. Kidd, M. C. Schnaidt, D. St. John, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J-K. Kim, C. Irvine, T. Levin, R. Wright, R. F. Freund, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: A Management System for Heterogeneous Networks," *8th IEEE Workshop on Heterogeneous Computing Systems (HCW '99)*, Apr. 1999, to appear.

[14] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18–27.

[15] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[16] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July–Sept. 1998, pp. 42–51.

[17] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, Vol. 220, No. 4598, May 1983, pp. 671–680.

[18] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. Webster, ed., John Wiley & Sons, New York, NY, to appear 1999.

[19] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[20] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, Jan. 1994, pp. 96–101.

[21] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[22] C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing system using a minimax criterion," *IEEE Transactions on Computers*, Vol. C-34, No. 3, Mar. 1985, pp. 197–203.

[23] P. Shroff, D. Watson, N. Flann, and R. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, April 1996, pp. 98–104.

[24] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.

[25] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[26] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17–26.

[27] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 1–15.

Biographies

Tracy D. Braun is a PhD student and Research Assistant in the School of Electrical and Computer Engineering at Purdue University. He received a Bachelor of Science in Electrical Engineering with Honors and High Distinction from the University of Iowa in 1995. In 1997, he received an MSEE from the School of Electrical and Computer Engineering at Purdue University. He received a Benjamin Meisner Fellowship from Purdue University for the 1995-1996 academic year. He is a member of IEEE, IEEE Computer Society, and Eta Kappa Nu honorary society. He is an active member of the Beta Chapter of Eta Kappa Nu at Purdue University, and has held several offices during his studies at Purdue, including chapter President. He has also been employed at Norand Data Systems and Silicon Graphics Inc./Cray Research. His research interests include parallel algorithms, heterogeneous computing, computer security, and software design.

H. J. Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received BS degrees in both Electrical Engineering and Management from MIT, and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Noah Beck is a Research Assistant and MSEE student at Purdue University in the School of Electrical and Computer Engineering. He received a Bachelor of Science in Computer Engineering from Purdue University in 1997, and is an active member of the Beta chapter of the Eta Kappa Nu honorary society. He has also been employed at Intel Corporation, and his research interests include microprocessor architecture, parallel computing, and heterogeneous computing.

Ladislau L. Bölöni is a PhD student and Research Assistant in the Computer Sciences Department at Purdue University. He received a Diploma Engineer degree in Computer Engineering with Honors from the Technical University of Cluj-Napoca, Romania in 1993. He received a fellowship from the Hungarian Academy of Sciences for the 1994-95 academic year. He is a member of ACM and the Upsilon Pi Epsilon honorary society. His research interests include distributed object systems, autonomous agents and parallel computing.

Muthucumaru Maheswaran is an Assistant Professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a BSc degree in Electrical and Electronic Engineering from the University of Peradeniya, Sri Lanka. He received an MSEE degree in 1994 and a PhD degree in 1998, both from the School of Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an MSEE student at Purdue University. His research interests include computer architecture, distributed computing, heterogeneous computing, Internet and World Wide Web systems, metacomputing, mobile programs, network computing, parallel computing, resource management systems for metacomputing, and scientific computing. He has authored or coauthored 15 technical papers in these and related areas. He is a member of the Eta Kappa Nu honorary society.

Albert I. Reuther is a PhD student and Research Assistant in the School of Electrical and Computer Engineering at Purdue University. He received his Bachelor of Science in Computer and Electrical Engineering with Highest Distinction in 1994 and received a Masters of Science in Electrical Engineering in 1996, both at Purdue. He was a Purdue Andrews Fellowship recipient in the 1994-95 and 1995-96 academic years. He is a member of IEEE, IEEE Computer Society, ACM, and Eta Kappa Nu honorary society and has been employed by General Motors and Hewlett-Packard. His research interests include multimedia systems, heterogeneous computing, parallel processing, and educational multimedia.

James P. Robertson currently works for Motorola's PowerPC System Performance and Modeling group. He received a Bachelor of Science in Computer Engineering with Honors from the school of Electrical and Computer Engineering at Purdue University in 1996. As an undergraduate student he received an NSF undergraduate research scholarship. In 1998 he received an MSEE from Purdue University. He is a member of IEEE, IEEE Computer Society, and Eta Kappa Nu honorary society. While attending Purdue

University he was an active member of the Beta Chapter of Eta Kappa Nu, having held several offices including chapter Treasurer.

Mitchell D. Theys is a PhD student and Research Assistant in the School of Electrical and Computer Engineering at Purdue University. He received a Bachelor of Science in Computer and Electrical Engineering in 1993 with Highest Distinction, and a Master of Science in Electrical Engineering in 1996, both from Purdue. He received support from a Benjamin Meisner Fellowship from Purdue University, an Intel Graduate Fellowship, and an AFCEA Graduate Fellowship. He is a member of the Eta Kappa Nu honorary society, IEEE, and IEEE Computer Society. He was elected President of the Beta Chapter of Eta Kappa Nu at Purdue University and has held several various offices during his stay at Purdue. He has held positions with Compaq Computer Corporation, S&C Electic Company, and Lawrence Livermore National Laboratory. His research interests include design of single chip parallel machines, heterogeneous computing, parallel processing, and software/hardware design.

Bin Yao is a PhD student and Research Assistant in the School of Electrical and Computer Engineering at Purdue University. He received Bachelor of Science in Electrical Engineering from Beijing University in 1996. He received Andrews Fellowship from Purdue University for the academic years 1996-1998. He is a student member of the IEEE. His research interests include distributed algorithms, fault tolerant computing, and heterogeneous computing.

Debra Hensgen is an Associate Professor in the Computer Science Department at The Naval Postgraduate School. She received her PhD in the area of Distributed Operating Systems from the University of Kentucky. She is currently a Principal Investigator of the DARPA-sponsored Management System for Heterogeneous Networks QUORUM project (MSHN) and a co-investigator of the DARPA-sponsored Server and Active Agent Management (SAAM) Next Generation Internet project. Her areas of interest include active modeling in resource management systems, network re-routing to preserve quality of service guarantees, visualization tools for performance debugging of parallel and distributed systems, and methods for aggregating sensor information. She has published numerous papers concerning her contributions to the Concurra toolkit for automatically generating safe, efficient concurrent code, the Graze parallel processing performance debugger, the SAAM path information base, and the SmartNet and MSHN Resource Management Systems.

Richard F. Freund is a founder and CEO of NOEMIX, a San Diego based startup to commercialize distributed computing technology. Dr. Freund is also one of the early pioneers in the field of distributed computing, in which he has written or co-authored a number of papers. In addition he is a founder of the Heterogeneous Computing Workshop, held each year in conjunction with IPPS/SPDP. Freund won a Meritorious Civilian Service Award during his former career as a government scientist.

```
initial population generation;
evaluation;
while (stopping criteria not met) {
        selection;
        crossover;
        mutation;
        evaluation;
}
```

Figure 1. General procedure for a Genetic Algorithm, based on [26].

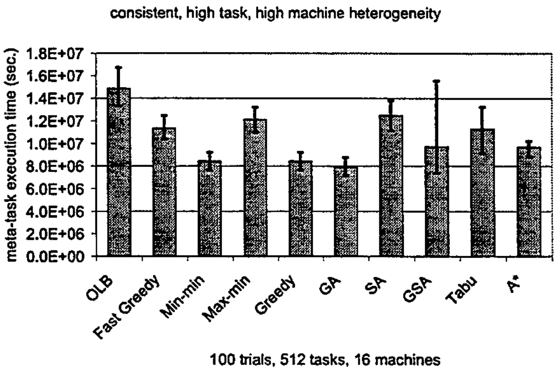


Figure 2. Consistent, high task, high machine heterogeneity.

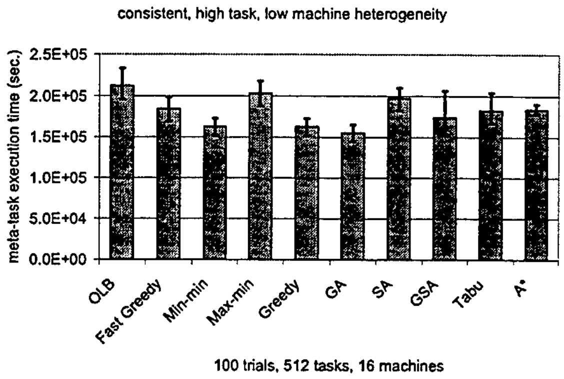


Figure 3. Consistent, high task, low machine heterogeneity.

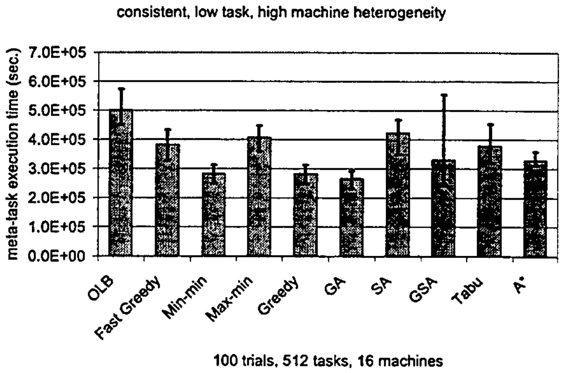


Figure 4. Consistent, low task, high machine heterogeneity.

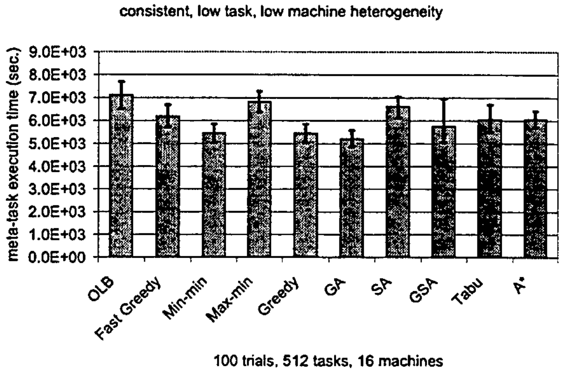


Figure 5. Consistent, low task, low machine heterogeneity.

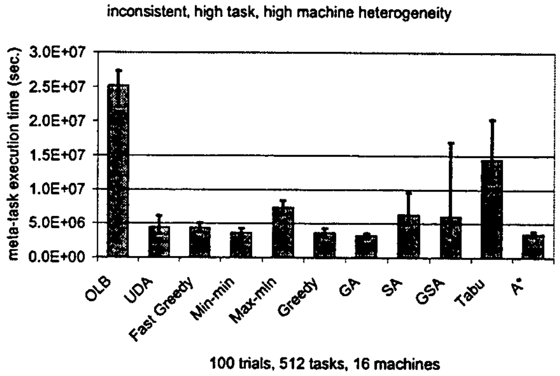


Figure 6. Inconsistent, high task, high machine heterogeneity.

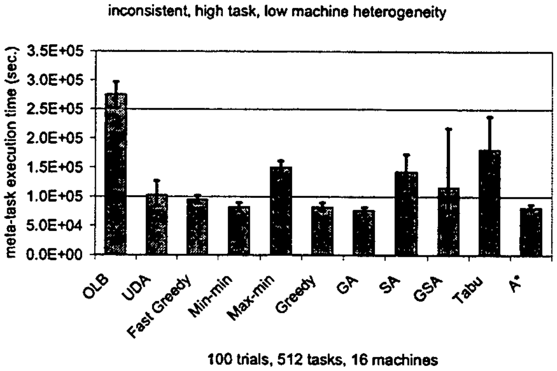


Figure 7. Inconsistent, high task, low machine heterogeneity.

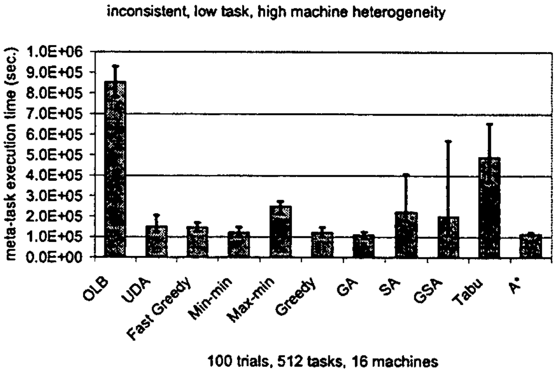


Figure 8. Inconsistent, low task, high machine heterogeneity.

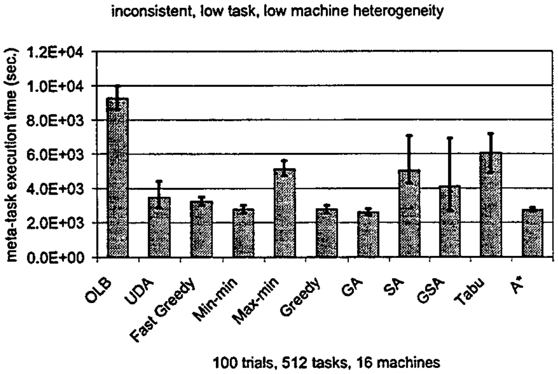


Figure 9. Inconsistent, low task, low machine heterogeneity.

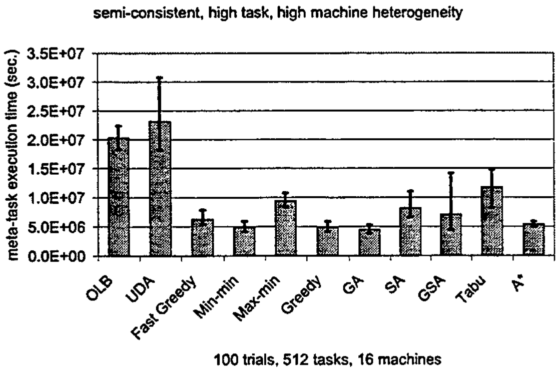


Figure 10. Semi-consistent, high task, high machine heterogeneity.

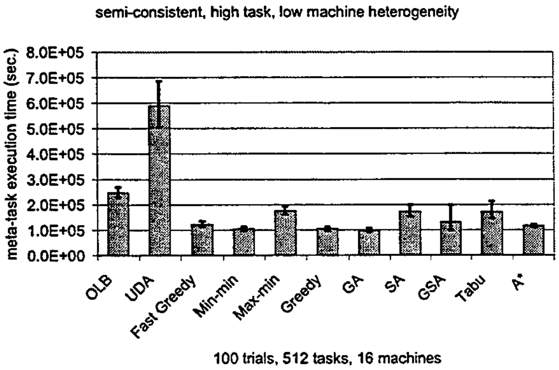


Figure 11. Semi-consistent, high task, low machine heterogeneity.

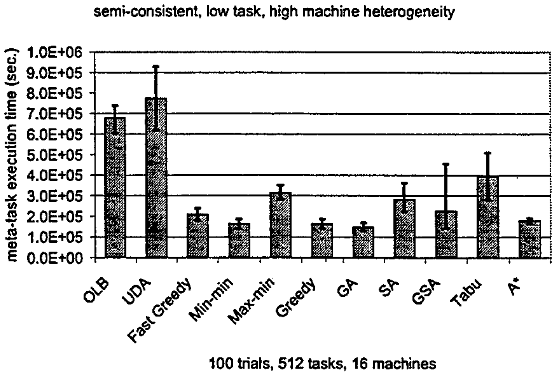


Figure 12. Semi-consistent, low task, high machine heterogeneity.

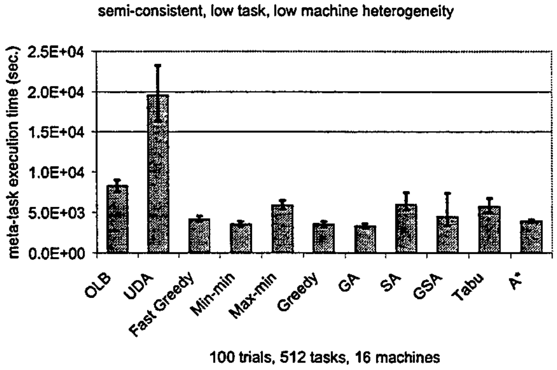


Figure 13. Semi-consistent, low task, low machine heterogeneity.

	machines							
t	436,735.9	815,309.1	891,469.0	1,722,197.6	1,340,988.1	740,028.0	1,749,673.7	251,140.1
a	950,470.7	933,830.1	2,156,144.2	2,202,018.0	2,286,210.0	2,779,669.0	220,536.3	1,769,184.5
s	453,126.6	479,091.9	150,324.5	386,338.1	401,682.9	218,826.0	242,699.6	11,392.2
k	1,289,078.2	1,400,308.1	2,378,363.0	2,458,087.0	351,387.4	925,070.1	2,097,914.2	1,206,158.2
s	646,129.6	576,144.9	1,475,908.2	424,448.8	576,238.7	223,453.8	256,804.5	88,737.9
	1,061,682.3	43,439.8	1,355,855.5	1,736,937.1	1,624,942.6	2,070,705.1	1,977,650.2	1,066,470.8
	10,783.8	7,453.0	3,454.4	23,720.8	29,817.3	1,143.7	44,249.2	5,039.5
	1,940,704.5	1,682,338.5	1,978,545.6	788,342.1	1,192,052.5	1,022,914.1	701,336.3	1,052,728.3

Table 1. Sample 8×8 excerpt from ETC with inconsistent, high task, high machine heterogeneity.

	machines							
t	21,612.6	13,909.7	6,904.1	3,621.5	3,289.5	8,752.0	5,053.7	14,515.3
a	578.4	681.1	647.9	477.1	811.9	619.5	490.9	828.7
s	122.8	236.9	61.3	143.6	56.0	313.4	283.5	241.9
k	1,785.7	1,528.1	6,998.8	4,265.3	3,174.6	3,438.0	7,168.4	2,059.3
s	510.8	472.0	358.5	461.4	1,898.7	1,535.4	1,810.2	906.6
	22,916.7	18,510.0	11,932.7	6,088.3	9,239.7	15,036.4	18,107.7	12,262.6
	5,985.3	2,006.5	1,546.4	6,444.6	2,640.0	7,389.3	5,924.9	1,867.2
	16,192.4	3,088.9	16,532.5	13,160.6	10,574.2	7,136.3	15,353.4	2,150.6

Table 2. Sample 8×8 excerpt from ETC with inconsistent, high task, low machine heterogeneity.

	machines							
t	16,603.2	71,369.1	39,849.0	44,566.1	55,124.3	9,077.3	87,594.5	31,530.5
a	738.3	2,375.0	5,606.2	804.9	1,535.8	4,772.3	994.2	1,833.9
s	1,513.8	45.1	1,027.3	2,962.1	2,748.2	2,406.3	19.4	969.9
k	2,219.9	5,989.2	2,747.0	88.2	2,055.1	665.0	356.3	2,404.9
s	12,654.7	10,483.7	10,601.5	6,804.6	134.3	10,532.8	12,341.5	5,046.3
	4,226.0	48,152.2	11,279.3	35,471.1	30,723.4	24,234.0	6,366.9	22,926.9
	20,668.5	28,875.9	29,610.1	7,363.3	24,488.0	31,077.3	8,705.0	11,849.4
	52,953.2	14,608.1	58,137.2	16,685.5	36,571.3	35,888.8	38,147.0	15,167.5

Table 3. Sample 8×8 excerpt from ETC with inconsistent, low task, high machine heterogeneity.

	machines							
t	512.9	268.0	924.9	494.4	611.2	606.9	921.6	209.6
a	8.5	16.8	23.4	19.2	27.9	22.7	19.6	8.3
s	228.8	238.5	107.2	180.0	334.6	88.2	192.8	125.7
k	345.1	642.4	136.8	206.2	559.5	349.5	640.2	664.2
s	117.3	235.9	149.9	71.5	136.6	363.6	182.8	359.5
	240.7	412.0	259.1	319.8	237.5	338.3	178.5	537.7
	462.8	93.3	574.9	449.4	421.8	559.6	487.7	298.7
	119.5	36.7	224.2	194.2	176.5	156.8	182.7	192.0

Table 4. Sample 8×8 excerpt from ETC with inconsistent, low task, low machine heterogeneity.

From the *IEEE Workshop on Advances in Parallel and Distributed Systems*, West Lafayette, IN, October 1998, pp. 330–335, (included in the *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998).

A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems

Tracy D. Braun[†], Howard Jay Siegel[†], Noah Beck[†], Ladislau Bölöni[‡], Muthucumaru Maheswaran[†],
Albert I. Reuther[†], James P. Robertson[†], Mitchell D. Theys[†], and Bin Yao[†]
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285 USA
[†]{tdbraun, hj, noah, maheswar, reuther, robertso, theys, yaob}@ecn.purdue.edu
[‡]boloni@cs.purdue.edu

Abstract

The problem of mapping (defined as matching and scheduling) tasks and communications onto multiple machines and networks in a heterogeneous computing (HC) environment has been shown to be NP-complete, in general, requiring the development of heuristic techniques. Many different types of mapping heuristics have been developed in recent years. However, selecting the best heuristic to use in any given scenario remains a difficult problem. Factors making this selection difficult are discussed. Motivated by these difficulties, a new taxonomy for classifying mapping heuristics for HC environments is proposed ("the Purdue HC Taxonomy"). The taxonomy is defined in three major parts: (1) the models used for applications and communication requests, (2) the models used for target hardware platforms, and (3) the characteristics of mapping heuristics. Each part of the taxonomy is described, with examples given to help clarify the taxonomy. The benefits and uses of this taxonomy are also discussed.

1. Introduction

Different portions of a computationally intensive application often have diverse computational requirements. In general, a high-performance machine may perform poorly on such an application because it is difficult for a single machine architecture to satisfy the computational requirements of the different portions equally well. A mixed-machine heterogeneous computing (HC) system alleviates this problem by utilizing a suite of different high-performance machines, interconnected with high-speed links. Such a system coordinates the execution of various portions of the application on different machines within the system to exploit the different architectural capabilities available and achieve increased application performance [12, 13].

To take advantage of HC systems in this manner, an application task may be decomposed into subtasks, where each subtask is computationally homogeneous. Different subtasks, however, may require different architectural capabilities. These subtasks may share stored or generated data, creating the potential for inter-machine dependencies and data transfer overhead. Subtasks may be determined by: (a) user specification, (b) analysis by the mapping heuristic, or (c) a given separate program for each subtask. Once the application is decomposed into subtasks, each subtask is assigned to a machine (matching) and the subtasks assigned to a particular machine are ordered (scheduling) such that the overall execution time of the application is minimized. The combination of matching and scheduling subtasks to machines is defined as subtask mapping. While each subtask is a separate item to be mapped, the mapping decision for any one subtask may impact the mapping decision for others.

Another version of the mapping problem, meta-task mapping, deals with matching and scheduling a collection of tasks to the machines in an HC environment. The term meta-task has been used in different ways. In this paper, the tasks in the meta-task are independent, in that they have no data dependencies among them. A given task, however, may have subtasks and dependencies among the subtasks. (In some systems, all tasks and subtasks in a meta-task, as defined above, are referred to as just tasks.) An example of meta-task mapping is the mapping of an arbitrary set of independent tasks from different users waiting to execute on a heterogeneous suite of machines. Each task in a meta-task may have associated properties, such as a deadline and a priority.

In general, finding optimal solutions for the map-

This research was supported in part by the DARPA/ITO Quorum Program under NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and in part by the DARPA/ISO BADD Program and the Office of Naval Research under grant number N00014-97-1-0804.

ping problem and the scheduling of inter-machine communications in HC environments is NP-complete [7], requiring the development of near-optimal heuristic techniques. In recent years, numerous different types of mapping heuristics have been developed (e.g., see [1, 6, 8, 12, 13]). However, selecting a particular heuristic to use in a certain practical scenario remains a difficult problem. One of the reasons for this difficulty is that when one heuristic is presented and evaluated in the literature, typically, different assumptions are made about the underlying target platform than those used for earlier heuristics, (e.g., the degree to which the capabilities of machines differ in the HC suite) making comparisons problematic. Similarly, different assumptions about application models complicate comparisons (e.g, the variation among average task execution times). Moreover, the mapping heuristics themselves usually have different characteristics (e.g., different optimization criteria, different execution times). Therefore, a fair comparison of various heuristics is a challenging problem.

These comparison problems are compounded by the fact that there exist no standard set of application benchmarks or target platforms for HC environments. Motivated by these difficulties, a new taxonomy for classifying mapping heuristics for HC environments is proposed. The Purdue HC Taxonomy is defined in three major parts: (1) the models used for applications and communication requests, (2) the models used for target hardware platforms, and (3) the characteristics of mapping heuristics. This new taxonomy builds on previous taxonomies (e.g., [2, 4, 5, 10]).

In Section 2, previous taxonomies from the fields of distributed computing and HC are reviewed. The proposed taxonomy for mapping heuristics is defined in Section 3. The benefits and possible uses of this new taxonomy are examined in Section 4.

This research was supported in part by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort among the Naval Postgraduate School, Noemix, Purdue University, and the University of Southern California. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver requested qualities of service. This research is also supported in part by the DARPA/ISO BADD (Battlefield Awareness and Data Dissemination) Program. In the BADD Program, communications from a large number of heterogeneous information sources (e.g., databases, sensors) to a large number of heterogeneous destinations (e.g., warfighters' laptops, proxy servers) must be scheduled over a set of heterogeneous networks [16]. Thus, most of this taxonomy pertains to this environment also.

2. Previous Taxonomies

Taxonomies related in various degrees to this work have appeared in the literature. In this section, overviews of three related taxonomy studies are given.

A taxonomy classifying scheduling techniques used in general-purpose distributed computing systems is presented in [2]. The classification of target platforms and application characteristics was outside the scope of this study. The taxonomy in [2] does combine well-defined hierarchical characteristics with more general flat characteristics to differentiate a wide range of scheduling techniques. Several examples of different scheduling techniques from the published literature are also given, with each classified by the taxonomy. In HC systems, however, scheduling is only half of the mapping problem. The matching of tasks to machines also greatly affects execution schedules and system performance. Therefore, the taxonomy proposed in Section 3 also includes categories for platform characteristics and application characteristics, both of which influence matching (and scheduling) decisions.

Several different taxonomies are presented in [5]. The first is the $\underline{EM}^3$ taxonomy, which classifies all computer systems into one of four categories, based on execution mode and machine model [4]. The taxonomy proposed here in Section 3 assumes heterogeneous systems from either the $SEMM$ (single execution mode, multiple machine models) or the $MEMM$ (multiple execution modes, multiple machine models) categories. A "modestly extended" version of the taxonomy from [2] is also presented in [5]. The modified taxonomy introduces new descriptors and is applied to heterogeneous resource allocation techniques. Aside from considering different parallelism characteristics of applications, target platform and application properties were not classified as part of the study.

A taxonomy for comparing heterogeneous subtask matching methodologies is included in [10]. The taxonomy focuses on static subtask matching approaches, and classifies several specific examples of optimal and sub-optimal techniques. This is a single taxonomy, without the three distinct parts of the Purdue HC Taxonomy presented in the next section. However, the "optimal-restricted" classification in [10] includes algorithms that place restrictions on the underlying program and/or multicomputer system.

The Purdue HC Taxonomy uses these studies as a foundation, and extends their concepts to the specific

HC mapping problem domain being considered. Relevant ideas from these studies are incorporated into the unique structure of the three-part taxonomy described in the next section, allowing for more detailed classifications of HC mapping heuristics.

3. Proposed Taxonomy

3.1. Introduction

As mentioned in Section 2, it is assumed that a mixed-machine HC system is composed of different machines, with possibly multiple execution models. The system is defined to be heterogeneous if any one or more of the following characteristics varies among machines enough to result in different execution performance among those machines: processor type, processor speed, mode of computation, memory size, number of processors (within parallel machines), interprocessor network (within parallel machines), etc.

The new Purdue HC Taxonomy for describing mapping heuristics for mixed-machine HC systems is defined by three major components: (1) application model and communication requests characterization, (2) platform model characterization, and (3) mapping strategy characterization. Previous taxonomies have focused only on the third item above. However, intelligent mapping decisions require information about both the hardware platform and the application being executed. Also, if there are special platform or application requirements (e.g., priorities associated with each task in a military environment), it is important that the mapping strategy be able to support these.

Thus, the Purdue HC Taxonomy classifies all three components of an HC environment, and attempts to *qualitatively* define aspects of the environment that can affect mapping decisions and performance. (Doing this *quantitatively* in a thorough, rigorous, complete, and "standard" manner is a long term goal of the HC field.) This taxonomy is based on existing mapping heuristics found in the literature, as well as previous research and experience within the field of HC.

Because research on mapping heuristics is an active and growing field, this taxonomy can only capture features of the current state of research at a certain level of detail. It is assumed that this taxonomy will be refined and expanded over time to serve as an evolving standard for describing HC mapping heuristics and their assumed environments.

3.2. Application model characterization

The first category of the taxonomy defines the *models* used for the applications to be executed on the HC system and for the communications to be scheduled on the inter-machine network. The applications themselves are not classified by functionality, but rather by the traits that define application computational characteristics that may impact mapping decisions. Furthermore, the taxonomy is able to include application traits that are subject to simplifying implementation assumptions (which may not reflect the most effective implementations), e.g., a subtask that is capable of beginning execution with a partial set of data is instead forced to wait until all input data arrives. The defining characteristics of the applications (which can be tasks or subtasks) are listed below. Many of the characteristics are also relevant to communication requests in BADD-like environments, including deadlines, mulitple versions, priorities, QoS requirements, and temporal distribution.

application size: How many tasks are in the meta-task and/or how many subtasks are in each task?

application type: What type of applications are to be mapped? If all tasks are independent, meta-task mapping is being performed. If there is a single task decomposed into subtasks (recall subtasks have dependencies), it is subtask mapping. One can also have the situation where a meta-task has independent tasks, but some of the tasks have subtasks. In this case, both meta-task and subtask mappings would be necessary.

communication patterns: What are the source and destination subtasks for each data item to be transferred?

data availability: The time at which input data needed by a subtask or output data generated by a subtask can be utilized varies in relation to subtask start and finish times: (a) is data available (to be forwarded) before a subtask completes, and (b) can a subtask begin execution before receiving all of its input data? As an example, the clustering non-uniform directed graph heuristic in [6] assumes that a subtask cannot send data to other waiting subtasks until it completely finishes executing.

deadlines: Do the applications have deadlines? This property could be further refined into hard and firm deadlines, if required. Applications completed by a firm deadline provide the most valuable results. An application that completes after a firm deadline but before a hard deadline is still able to provide some useful data. After a hard deadline has passed, data from the application is useless.

execution time model: Most mapping techniques require an estimate of the execution time of each application on each machine. How are the estimated execution times modeled? The two choices most commonly used are probabilistic and deterministic modeling. Probabilistic modeling uses a probability distribution for application execution times when making mapping decisions [1, 11]. Deterministic modeling uses a fixed (or expected) value [8], e.g., the average of ten previous executions of an application.

meta-task heterogeneity: For each machine in the HC suite, how greatly and with what properties (e.g., probability distribution) do the execution times of the different tasks in the meta-task vary?

multiple versions: Do the applications have multiple versions that could be executed? For example, an application that requires an FFT might be able to perform the FFT with either of two different procedures that have different precisions, different execution times, and different resource requirements. What are the relative "values" of the different versions to the user?

priorities: Do the applications have priorities? Environments that would require priorities include military systems and machines where time-sharing must be enforced. Priorities are generally assigned by the user (within some allowed range), but the relative weightings given to each priority are usually determined by another party (e.g., a system administrator). Priorities and their relative weightings are required if the mapping strategy is preemptive (Subsection 3.4).

QoS requirements: Certain application specific Quality of Service (QoS) requirements may need to be considered, such as security level.

subtask heterogeneity: Similar to meta-task heterogeneity above.

task profile: Task profiling specifies the types of computations present in an application based on the code for the task (or subtask) and the data to be processed [9, 13]. This information may be used by the mapping heuristic, in conjunction with analytical benchmarking (Subsection 3.3), to estimate task (or subtask) execution time.

temporal distribution: Is the complete set of tasks of a meta-task to be mapped known *a priori* (static applications), or do the tasks arrive in a real-time, non-deterministic manner (dynamic applications), or is it a combination of the two?

Because the characteristics defined above are largely independent of each other, these would all be considered flat characteristics in a taxonomy, not hierarchical characteristics with dependencies. Each of the previous taxonomies listed in Section 2 used a hierarchical structure to show relationships. The "checklist" format above is necessary to capture all of the aspects of applications that can influence mapping decisions.

3.3. Platform model characterization

The second category of the taxonomy defines the models used for target hardware platforms available within HC systems. Several existing heuristics make simplifying (but unrealistic) assumptions about their target platforms (e.g., [14] assumes an infinite number of machines are available). Therefore, this taxonomy is not limited to a set of realistic target platforms. Instead, a framework for classifying the *models* used for target platforms is provided below.

analytical benchmarks: Analytical benchmarking provides a measure of how well each available machine in the HC platform performs on each given type of computation [9, 13]. This information may be used by the mapping heuristic, in conjunction with task profiling (Subsection 3.2), to estimate task (or subtask) execution time.

communication time: How much time does it take to send data from any one machine to any other? This may be expressed as a function of path establishment time and bandwidth.

concurrent send/receives: Can each machine perform concurrent sends and receives to other machines (assuming enough network connections)?

interconnection network: Volumes of literature already exist on the topic of interconnection networks, therefore, they are not classified here. (A general interconnection network taxonomy can be found in [3].) It is merely noted that many network characteristics can affect mapping decisions and system performance, including the following: bandwidth, ability to perform concurrent data transfers, latency, switching control, and topology. Most of these network properties are also functions of the source and destination machines.

machine architecture: For each machine, various architectural features that can impact performance must be considered, e.g., processor type, processor speed, external I/O bandwidth, mode of computation (e.g., SIMD, MIMD, vector), memory size, number of processors (within parallel machines), and interprocessor network (within parallel machines).

machine heterogeneity: For each task (or subtask), how greatly and with what properties (e.g., probability distribution) do the execution times for this task vary across different machines in the HC suite?

number of connections: How many connections does each machine have to the interconnection network structure or directly to other machines?

number of machines: This property is defined by two subclasses, based on the quantity and variability of the number of machines: (a) finite or infinite, and (b) fixed or variable (e.g., new machines can come on-line). Furthermore, a given heuristic with a finite, fixed number of machines may treat this number as a parameter that can be changed from one mapping to another.

overlapped computation/communication: Can machines overlap computation and inter-machine communication?

system control: Does the mapping strategy control and allocate all resources in the environment (dedicated), or are external users also consuming resources (shared)?

task compatibility: Is each machine in the environment able to perform each application, or, for some applications, are special capabilities that are only available on certain machines required? These capabilities could involve issues such as I/O devices, memory space, and security.

3.4. Mapping strategy characterization

The third category of the Purdue HC Taxonomy defines the characteristics used to describe the mapping strategies. Because the general HC mapping problem is NP-complete, it is assumed that the mapping strategies being classified are near-optimal techniques.

application model supported: See Subsection 3.2.

communication times: Are inter-subtask data communication times considered during subtask mapping?

control location: Is the mapping strategy centralized or distributed? Distributed strategies can further be classified as cooperative or non-cooperative (independent) approaches.

data forwarding: Is data forwarding considered during mapping [15]? That is, could a subtask executing on a machine receive data from an intermediate machine sooner than from the original source?

dependencies: This property is closely related to the application type from Subsection 3.2. Meta-task mapping deals with an independent set of tasks. Subtask mapping handles the case where there are one or more tasks with subtasks and dependencies.

duplication: Can a given subtask be duplicated and executed on multiple machines to reduce communication overhead?

dynamic/static: Dynamic mapping techniques operate during (and possibly before) application execution time, and make use of real-time information. Dynamic techniques require inputs from the environment, and may not have a definite end. For example, dynamic techniques may not know the entire set of tasks to be mapped when the technique begins executing; new tasks may arrive at random intervals. Similarly, new machines may be added to the suite. If a dynamic technique has feedback, applications may be reassigned because of the loss of a machine, or application execution times taking significantly longer than expected. In contrast, static mapping techniques take a fixed set of applications, a fixed set of machines, and a fixed set of application and machine attributes as inputs and generate a single, fixed mapping. Static mapping techniques have a well-defined beginning and end, and each resulting mapping is not modified due to changes in the HC environment or feedback.

execution location: Can a machine within the suite be used to execute the mapping strategy, or is an external machine required?

execution times: The execution times of different mapping strategies vary greatly, and are an important property during the comparison or selection of mapping techniques. Can the execution time of the heuristic accurately be predicted, e.g., does the mapping heuristic perform a fixed, predetermined number of steps (e.g., greedy approaches [1]) before arriving at a mapping, or is the heuristic iterative in the sense that the mapping is continually refined until some stopping criteria is met, resulting in a number of steps that is not known *a priori* (e.g., genetic algorithms [13, 14])?

fault tolerance: Is fault tolerance considered by the mapping strategy? This may take several forms, such as assigning applications to machines that can perform checkpointing, or executing multiple, redundant copies of an application.

feedback: Does the mapping strategy incorporate real-time feedback from the platform (e.g., machine availability times) or applications (e.g., actual task execution times) into its decisions? Strategies that utilize feedback are dynamic, but not all dynamic strategies have feedback.

objective function: The quantity that the mapping strategy is trying to optimize. This varies widely between strategies, and can make some approaches inappropriate in some situations.

platform model supported: See Subsection 3.3.

preemptive: Preemptive mapping strategies can interrupt applications that have already begun execution, to free resources for more important applications. Applications that were interrupted may be reassigned, or may resume execution upon completion of the more important application. Preemptive techniques must be dynamic by definition. Application "importance" must be specified by some priority assignment and weighting scheme, as discussed in Subsection 3.2.

remapping: Does the mapping heuristic require an initial mapping, which it then enhances? For example, a dynamic heuristic with feedback can remap a previous static mapping [13].

4. Conclusions

The mapping of tasks and meta-tasks, and the scheduling of communications, in HC environments are active, growing areas of research. Based on existing mapping approaches in the literature, a three-part taxonomy was proposed. The Purdue HC Taxonomy classified characteristics within the application models, target hardware platform models, and mapping strategies that are used in HC research. By defining all three categories, heterogeneous mapping techniques can more accurately be classified.

The Purdue HC Taxonomy can be beneficial to researchers in several ways. Currently, it is difficult to meaningfully compare different mapping approaches. Similarly, it is difficult to extend existing work or recognize important areas of research without understanding the relationships that exist among previous efforts. The three-part classification system provided allows HC researchers to more easily describe mapping heuristics, and to see design and environment alternatives, during the development of new heuristics, they might not have otherwise considered. A researcher can also use the taxonomy to find the mapping heuristics that use similar target platform and application models. The mapping heuristics found for similar models can then possibly be adapted or developed further to better solve the mapping problem that is being considered. In the future, this taxonomy could focus research towards the development of a standard set of benchmarks for HC environments. It is expected, as research progresses, that the Purdue HC Taxonomy will be an evolving standard, that is refined and extended to incorporate new ideas and findings.

Acknowledgments - The authors thank Ishfaq Ahmad, Shoukat Ali, Richard F. Freund, Veljko Milutinović, and Janet Siegel for their comments.

References

[1] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *7th Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79–87.

[2] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, Feb. 1988, pp. 141–154.

[3] J. Duato, S. Yalmanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[4] I. Ekmečić, I. Tartalja, and V. Milutinović, "A taxonomy of heterogeneous computing," *IEEE Computer*, Vol. 28, No.12, Dec. 1995, pp. 68–70.

[5] I. Ekmečić, I. Tartalja, and V. Milutinović, "A survey of heterogeneous computing: Concepts and systems," *Proceedings of the IEEE*, Vol. 84, No. 8, Aug. 1996, pp. 1127–1144.

[6] M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[7] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[8] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multiuser, heterogeneous, computing environments with SmartNet," *7th Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184–199.

[9] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, Jun. 1993, pp. 78–86.

[10] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous computing systems," *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 135–146.

[11] Y. A. Li and J. K. Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 172–184.

[12] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.

[13] H. J. Siegel, M. Maheswaran, and T. D. Braun, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. Webster, ed., John Wiley & Sons, New York, NY, to appear.

[14] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[15] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling subtasks and communication traffic in a heterogeneous computing system," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, Aug. 1997, pp. 857–871.

[16] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk, "A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment," *7th Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 115–129.

Are CORBA Services Ready to Support Resource Management Middleware for Heterogeneous Computing? *

Alpay Duman, Debra Hensgen, David St. John, and Taylor Kidd

Computer Science Department
Naval Postgraduate School
Monterey, CA 93940

Abstract

The goal of this paper is to report our findings as to which CORBA services are ready to support distributed system software in a heterogeneous environment. In particular, we implemented intercommunication between components in our Management System for Heterogeneous Networks (MSHN[1]) using four different CORBA mechanisms: the Static Invocation Interface (SII), the Dynamic Invocation Interface (DII), Untyped Event Services, and Typed Event Services. MSHN's goals are to manage dynamically changing sets of heterogeneous adaptive applications in a heterogeneous environment. We found these mechanisms at various stages of maturity, resulting in some being less useful than others. In addition, we found that the overhead added by CORBA varied from a low of 10.6 milliseconds per service request to a high of 279.1 milliseconds per service request on workstations connected via 100 Mbits/sec Ethernet. We therefore conclude that using CORBA not only substantially decreases the amount of time required to implement distributed system software, but it need not degrade performance.

1 Introduction

This paper describes the experiences we had using CORBA mechanisms to implement intercommunication in MSHN. MSHN's goal is to support the execution of multiple, disparate, adaptive applications[2] in a dynamic, distributed heterogeneous environment. To accomplish this goal, MSHN consists of multiple, distinct, and eventually replicated distributed components that themselves execute in a heterogeneous environment. These components have widely varying functionality, come in and out of existence, and communicate across heterogeneous networks. In addition to executing on different types of platforms, these components are also likely to be written in different programming languages. We can, of course, at the expense of a great deal of programmer's time, implement specialized naming services to locate the appropriate component at run-time, and specialized communication mechanisms to enable communication between the heterogeneous platforms upon which the components run. Alternatively, we can use a general tool, such as the Common Object Request Broker Architecture (CORBA), to achieve the same functionality while reducing our development time. Experience with generalized systems, such as CORBA, has revealed that the reduction in development time costs come at the expense of run-time performance, which can be critical in real-time applications. This research, therefore, investigates the utility and overhead of communication mechanisms, which are implemented according to the CORBA 2.2 specification, to support MSHN's inter-component communication.

We note to the reader that our interest lies in the CORBA mechanisms that support the development of (possibly real-time) resource management environments. This is a very specific realm where system overheads can have a significant impact on performance. We do not explore the many and varied capabilities of CORBA for the supporting of other environments, such as that of distributed general database services and video streaming. Our interest in CORBA is primarily as a tool to reduce the time/programming investment needed to implement our resource management system middleware. As the services and mechanisms provided by the CORBA 2.2 specification, particularly Static and Dynamic Invocation, and the Event Services, hold great promise in this regard, we performed the series of studies detailed in this paper.

CORBA specifies a standard to permit different pro-

*This research was supported by DARPA under contract number E583. Additional support was provided by the Naval Postgraduate School and the Institute for Joint Warfare Analysis.

[1] Pronounced "mission"

[2] This paper focuses on the use of CORBA mechanisms to support the components of MSHN, not the applications that MSHN itself supports. For more details concerning applications, please see the references or contact the the authors directly.

grams, executing on different computers, to request services from one another. CORBA's Naming Service and Object Request Brokers (ORBs) aid clients in locating appropriate servers. CORBA's static invocation enables a CORBA client to make a request of a server that is identified prior to compile time. It provides both reliable synchronous semantics and unreliable asynchronous semantics. In contrast, CORBA's dynamic invocation enables the client to locate a server that may not be known until run-time, and provides reliable synchronous and asynchronous semantics, as well as unreliable asynchronous semantics. CORBA's event services allow processes on one machine to place event notifications intended for processes on other machines into event queues so that the notifications can later be delivered to the serving processes. This service facilitates multicast. This paper will not cover CORBA in detail, but there are many other good references on the subject [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

The paper is organized as follows. We first briefly describe MSHN, concentrating on the type of intercommunication that is required by its components. A more complete description of MSHN can be found elsewhere [12]. Alternate designs for facilitating communication within MSHN itself and the implementation of these designs are presented. These designs are based upon, respectively, static invocation, dynamic invocation, untyped event service and typed event service. In this section, we also provide a qualitative assessment detailing the problems that we encountered while attempting to use these mechanisms within MSHN. In a subsequent section, we describe our experiments for evaluating these mechanisms within MSHN and present a quantitative analysis of each of the mechanisms. Finally, we summarize our findings.

2 The Management System for Heterogeneous Networks (MSHN)

In the Heterogeneous Processing Laboratory at the Naval Postgraduate School, we are designing, implementing, and testing a resource management system called the Management System for Heterogeneous Networks (MSHN). MSHN is designed as a general experimental platform for investigating issues relating to the design and construction of future resource management systems operating in heterogeneous environments. Though MSHN is used to explore a large number of such issues, our present research focuses on finding and developing (1) mechanisms for supporting adaptive applications, (2) mechanisms for supporting the satisfaction of user and system defined Quality of Service (QoS) requirements, and (3) mechanisms for acquiring and usefully aggregating measurements of both general resource availability and the resource usage of individual tasks. A thorough and complete description of MSHN can be found in Hensgen [12].

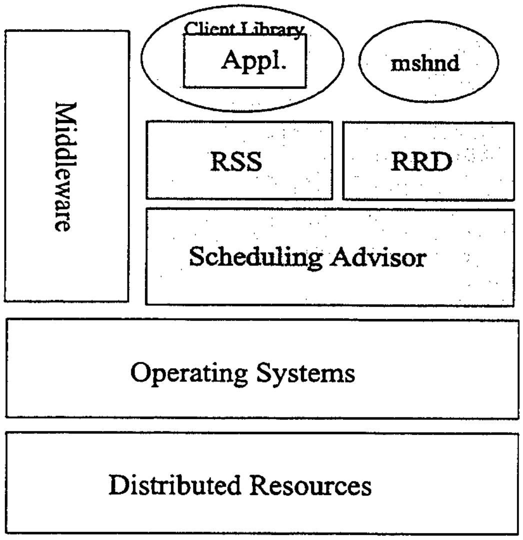


Figure 1: MSHN Conceptual Architecture

MSHN's architecture consists of multiple instantiations of each of the components enumerated below:

- a Client Library (one for each executing application to be managed by MSHN),
- a Scheduling Advisor (hierarchically replicated),
- a Resource Requirement Database (hierarchically replicated),
- a Resource Status Server (hierarchically replicated), and
- a MSHN Daemon (when needed).

Figure 1, the MSHN Conceptual Architecture, shows all of the MSHN components (shaded) as **translucent layers** executing on distributed platforms. A translucent layer is one that can be bypassed by layers that are above or below it. For example, the MSHN Daemon (mshnd) can interact directly with the operating systems layer, bypassing the Resource Status Server, the Resource Requirement Database and the Scheduling Advisor. In the environment that MSHN supports, both MSHN and non-MSHN applications may be executing at any given time. Figure 2 illustrates how these components, along with various MSHN and non-MSHN applications, might actually be distributed among different heterogeneous machines.

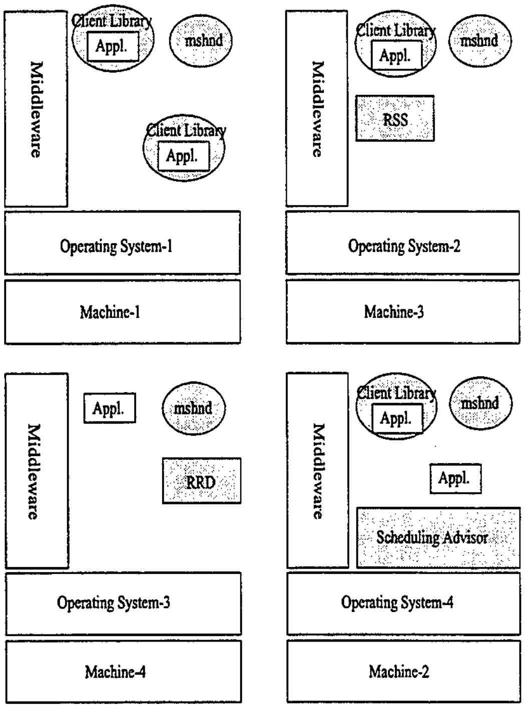


Figure 2: Example MSHN Physical Instantiation

This research investigates how communication between the components can be facilitated. As such, the MSHN description in the remainder of this section emphasizes that communication.

Figure 3, MSHN's Software Architecture, illustrates all of the interactions between the components. MSHN has a peer-to-peer architecture[3].

We now present two- and three-tier views to give a clear understanding of the interactions between the components. Generally, many applications, each linked with the MSHN Client Library, will be running at any given time. They will need to communicate with a Scheduling Advisor (SA) to request the appropriate resources needed to start new processes. They may also communicate with a MSHN Daemon when receiving their recommended schedule. Additionally, their Client Libraries update the Resource Requirement Database (RRD) and the Resource Status Server (RSS) with the expected resource requirements of the applications and current resource availability within the MSHN system.

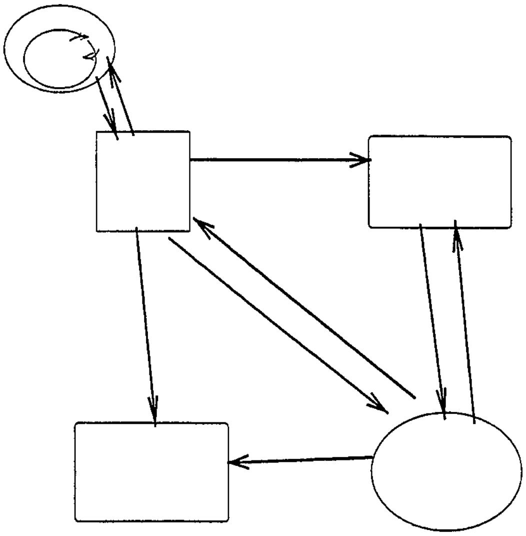


Figure 3: MSHN's Software Architecture

Figure 4 illustrates this updating interaction as a two-tiered client/server architecture. The arrows labeled "1" designate the Resource Requirements Database update path, and those labeled "2," the Resource Status Server update path. The update frequency of the Resource Status Server is expected to be high so that it, in turn, can supply the Scheduling Advisor with accurate and current information.

We anticipate that the frequency of the updates will load down the network, and cause a considerable processing load on the Resource Status Server and the Resource Requirement Database. To avoid these loads, MSHN's design includes proxy Resource Status Servers and Resource Requirement Databases that will come in and out of existence as required to minimize the number of updates. These proxies will filter gathered information and update the hierarchical Resource Status Server and the hierarchical Resource Requirement Database when necessary.

In one view, the Scheduling Advisor functionally resides between the information needed to create a schedule (the Resource Status Server and the Resource Requirement Database) and the requesters of schedules (applications linked with the Client Library). This indicates that there will be a high communication rate to and from the Scheduling Advisor. We can therefore also view MSHN as having three tiers, where the Scheduling Advisor is the second tier, and the Resource Status Server and the Resource Requirement Database are in the third tier (see Figure 5). When the Client Library (first tier) contacts the Scheduling Advisor for

[3] When callbacks are used the client and the server have a peer-to-peer relationship. In distributed systems, callbacks are useful as a mechanism for performing asynchronous communication. Callbacks transmit event notifications without blocking the event originator. Callbacks flow from the servers towards the clients.

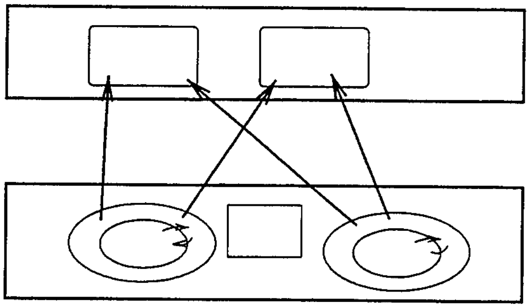


Figure 4: Two-tiered Architectural View of MSHN Architecture

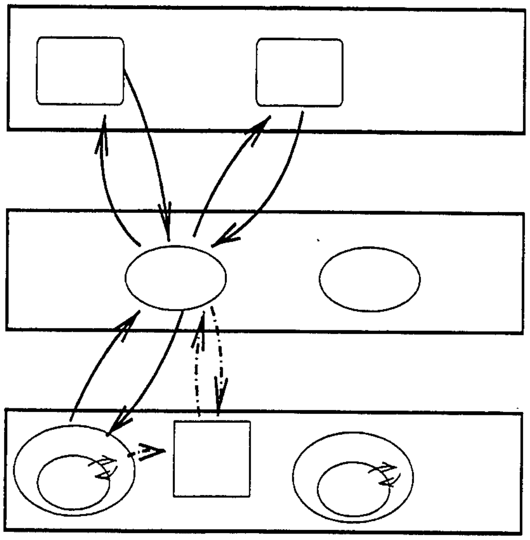


Figure 5: Three-tiered View of MSHN

a schedule, either directly or via the MSHN Daemon (the arrows labeled "1" and "1a"), the Scheduling Advisor queries both the Resource Status Server (arrows "2" and "3"), and the Resource Requirement Database (arrows "4" and "5") before it computes its schedule and sends it to the MSHN Daemon or client library depending upon which is more appropriate (arrows "6" and "6a")

Although the Client Libraries are the initiators of many of the communication chains through the MSHN system, other chains are initiated by the Resource Status Server. For example, in the case where a violation of a deadline occurs because of a change in resource availability, the Resource Status Server will trigger the Scheduling Advisor to reschedule processes that would not otherwise meet their deadline. The Scheduling Advisor will adapt to the new situation by either changing

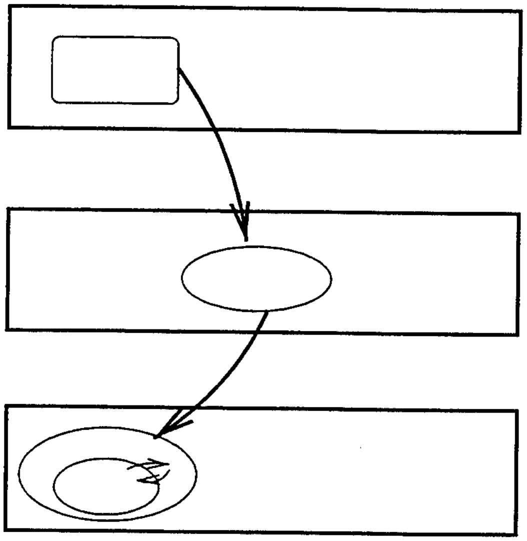


Figure 6: Alternate Three-tiered View of MSHN

the format[4] of the process or restarting it on a different resource, possibly via the MSHN Daemon. This interaction is the reverse of the previously described communication chain and can be used to define another version of a three-tiered view. (See Figure 6.)

Although we have shown several two and three tier views of MSHN, the reader should understand that these are only examples. Much larger chains will actually exist when the various components are hierarchically replicated.

3 Use of CORBA Services in MSHN and Problems Encountered

Our goal is to determine both (1) how we can best facilitate efficient communication between the components in our architecture using mechanisms from the CORBA 2.2 specification, and (2) to determine the runtime overhead of each of those mechanisms. Our justification for choosing a particular mechanism included extensibility, scalability, portability, flexibility, and efficiency.

MSHN consists of multiple, eventually replicated, distinct distributed components that execute in a heterogeneous environment. These components will have widely varying functionality, will come in and out of existence, will communicate via heterogeneous networks, and will execute on different platforms. To facilitate the interactions between MSHN's components, we identified four mechanisms from the CORBA 2.2 specifica-

[4] We use the term "format" to refer to a mechanism we have developed to support adaptive applications [13].

tion that had particular promise: the Typed Event Service, the Untyped Event Service, the Static Invocation Interface (SII), and the Dynamic Invocation Interface (DII). After settling on these four mechanisms, we implemented a prototype of MSHN's communication infrastructure using each of them. First we describe how the MSHN architecture would benefit from the both the Typed and Untyped Event Service, the Static Invocation Interface (SII), and the Dynamic Invocation Interface (DII). Then we discuss how we use the Naming Service within MSHN to obtain object references. In this section, since part of the objective of this paper is to make recommendations with regards to additions and improvements to the evolving CORBA specification, we describe and justify each of our designs, the problems we encountered, and the solutions to which we arrived.

3.1 Selection of a CORBA ORB

At the beginning of this research, we explored various implementations of the CORBA standard. Figures 7 and 8 present a summary of the results of that exploration[5]. Based upon various requirements, including the cost of some of the implementations, the time required to implement comparative tests, and the duration of this study, we had to limit ourselves to one CORBA implementation. We chose the implementation that seemed, at that time, to have the most mature features relevant to MSHN. Our assumption was that once such an implementation was found, other implementations would typically have similar difficulties and comparable performance. As such, we based our studies around IONA's Orbix, the implementation that best fit this requirement.

3.2 Event Service

Event Service allows multiple suppliers and multiple consumers to deliver and receive notifications for a set of events. An Event Channel transparently permits (1) suppliers to send notifications of events and (2) consumers to receive these notifications, all without knowledge of the existence of one another. Hence, the Event Service will support the transparent replication of MSHN system components for reliability and dependability. Event Service will enable Client Libraries, linked with different concurrent applications, to communicate with other MSHN components seamlessly. Finally, Event Service supports a standard Application Programming Interface (API) (e.g., for the Push-Push Model, a single operation push() taking a variable of type any as a parameter) which eases the development of MSHN system components.

Though there are four models for Event Service, there were only two available in relatively robust industrial implementations when we performed our experiments: the Push-Push Model and the Pull-Pull Model [14]. Using the Pull-Pull Model creates an additional load on the consumers. Because our servers, the consumers in this case, must minimize their use of computing resources even when there is no event to be delivered on the Event Channel, we chose to use only the Push-Push Model.

3.2.1 Using Event Service in MSHN

Figure 9 illustrates the use of Event Service to organize communication in the MSHN architecture. In this approach, the components of MSHN must register themselves as both a consumer and a supplier to the Event Channel. The Event Channel acts as the glue between all of the components and delivers notifications to each of them.

3.2.2 Problems with Initial Approach

Although this approach helps to organize MSHN's communication, providing transparent reliability and scalability, some problems can be seen involving both performance and the CORBA 2.2 specification. Some of the problems with this approach are identical to the problems identified by Schmidt and Vinoski in the analysis of their stock market application [11]. We first summarize their findings in the first two items below, Loss of Events in the System, and Problems with the Untyped Event Service. Then we enumerate additional problems that are particular to using CORBA within the MSHN architecture. Lastly, we look at how to implement a component that is both a supplier and a consumer.

Loss of Events in the System. Event Service guarantees delivery of notifications to all registered consumers as long as the Event Service process does not fail[6]. However, in the Event Service specification, persistency of events in the Event Channel is not required. Therefore, if an Event Service process does fail, undelivered notifications in the system may be lost.

The loss of notifications is fatal for MSHN because we are creating an environment for mission-critical applications. The obvious solution to this problem is to

[5] The capabilities of the various implementations of CORBA evolve very quickly. The content of these figures present the state of some of the implementations at the time this research was performed. As the capabilities of most CORBA implementations can quickly change, the reader is recommended to do his own similar exploration.

[6] Although there are many definitions of failure, we specifically mean that if the Event Service does not fail, then all consumers receive the correct value. This agrees with Lamport's definition of failure [15].

Vendor	Naming	Life Cycle	Event	Trading	Identity	Relationships
Expersoft	yes		yes			yes
Sun	yes	yes	yes		yes	yes
IONA	yes		yes	yes		
Visigenic	yes		yes			
BEA						
ICL			yes			
HP	yes	yes	yes	yes		
IBM	yes	yes	yes		yes	
Chorus						
OOT	yes			yes		
Electra	yes	yes	yes			
Xerox						
BBN	yes	yes				

Figure 7: Available Services

Vendor	Concurrency	Externalization	Persistency	Transactions	Security
Expersoft					
Sun					
IONA				yes	
Visigenic				yes	
BEA					yes
ICL				yes	yes
HP				yes	
IBM	yes	yes	yes	yes	
Chorus					
OOT		yes			
Electra					
Xerox					
BBN			yes		

Figure 8: Available Services (Continued)

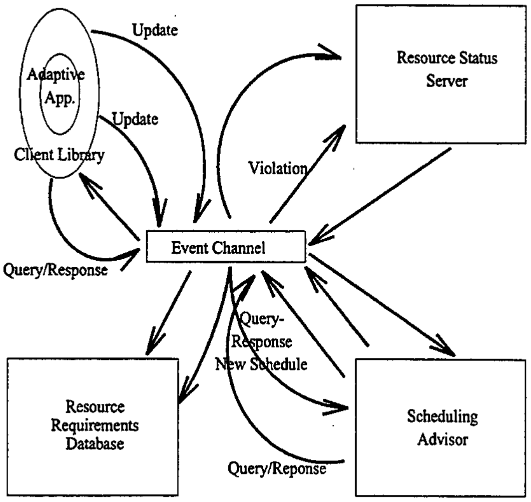


Figure 9: Using Event Service in MSHN

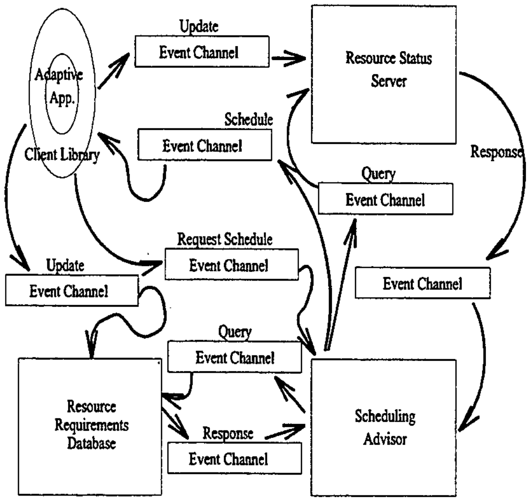


Figure 10: Using UntypedEvent Service

redefine the Event Service specification to include persistency for the undelivered notifications in the Event Channel. The OMG has been defining this requirement in the Notification Service specification [7]. However, no vendors had implemented this new specification at the time of this research.

Problems with Untyped Event Service. The Untyped Event Service does not specify any way to filter notifications. Therefore when using this service, all notifications are received by all registered consumers.

Passing all of these notifications in MSHN, many of which will be discarded by any particular consumer, through the network will increase the network load between the Event Channel and the consumer. Additionally, the consumers must filter events and convert the parameters that have type any to the type that is expected. In this case, there is an additional and unwanted load on the consumers to process all the events received. Finally, when more suppliers, in particular more applications, register with the Untyped Event Channel, more events will be generated in the system. Since the Untyped Event Channel delivers each event to all of the registered consumers and the consumers will filter all the events, the network load and consumer load will increase rapidly.

To handle this problem, we can use Typed Event Channels which filter the notifications according to their type. With this solution, the consumers receive only the notifications for which they register, decreasing the network traffic. In this solution, one Event Channel processes all of the notifications and delivers them only to the corresponding consumers. This also lightens the loads on the consumers because they avoid having to examine and discard events not meant for them. However, we note that it increases the computational load on the Event Channel. Later, we compare the run-time performance of Typed Event Channel to Untyped Event Channel using this approach in the MSHN architecture.

Alternatively, since we only have five different types of components in MSHN, we could use different channels for each connection between these components. In this approach, each Event Channel will only support one notification type. For example, for the Client Library - Scheduling Advisor Event Channel, we will have the Client Library as a supplier, the Scheduling Advisor as a consumer, and the possible client scheduling requests as the types of the notifications. Each MSHN component may be replicated by registering the additional (identical) components to the same Event Channel. This solution is shown in Figure 10.

Obviously, some combination of these two solutions may be best. That is, the Typed Event Channel itself can become a bottleneck in the first solution. Therefore, replication of Typed Event Channels may better fit MSHN's requirements. In this paper, we focused on the careful analysis of individual solutions rather than empirically exploring the exponentially sized solution space that combining these two techniques will create.

How to implement a component that is both a supplier and a consumer in a system in order to minimize the run-time overhead. All components of MSHN are both consumers and suppliers. Also, and perhaps particular to MSHN, when a component receives a notification, it usually becomes a supplier by generating another notification and delivering it to the appropriate Event Channel. Figure 11 shows the process of passing notifications from the Client Library to the Scheduling Advisor using the **push()** operation. It reveals how the Scheduling Advisor changes from a consumer to a supplier. In the Untyped Event Service's Push-Push Model, the supplier (here the Client Library) invokes a default **push()** operation on the Event Channel which in turn invokes a **push()** operation supplied by the developer of the consumer (here the Scheduling Advisor). In the **push()** operation that the developer supplied for the Scheduling Advisor (as a consumer), the developer of the Scheduling Advisor invokes the default push operation on the Scheduling Advisor – Resource Requirement Database (SA – RRD) Event Channel (which of course, invokes the **push()** operation supplied by the developer of the Resource Requirements Database).

The design issue here is to determine how to supply the Interoperable Object Reference (IOR) of the SA – RRD Event Channel to the **push()** operation of the Scheduling Advisor. We want to avoid using the Naming Service every time the **push()** operation (here the push operation of the Scheduling Advisor) is invoked. Instead, the developer can locate the SA–RRD Event Channel in the servant implementation. That is, the servant implementation will obtain the IOR for the SA–RRD Event Channel, stringify the IOR, and storing it in a file. The **push()** operation implementation can retrieve these IORs from their files, as needed, and deliver generated events, thereby pushing the corresponding notifications to the channel.

Therefore in the Untyped Event Service, to react to the notification (here a request for a schedule) that the consumer receives, the developer of the consumer (here the Scheduling Advisor) must override the default **push()** operation between the Event Channel and the consumer. For example, when the Scheduling Advisor receives an event from the Client Library requesting a schedule, it will generate a query notification for the Resource Requirement Database and deliver it to the SA – RRD Event Channel. In this case, the Scheduling Advisor becomes a supplier and is required to locating the SA – RRD Event Channel. To avoid locating the Event Channel to which the supplier will deliver the notification, via the Naming Service inside the **push()** operation, the developer can locate the Event Channel in the servant implementation and obtain IORs of it. Then, the servant implementation can stringify these IORs and store them in files.

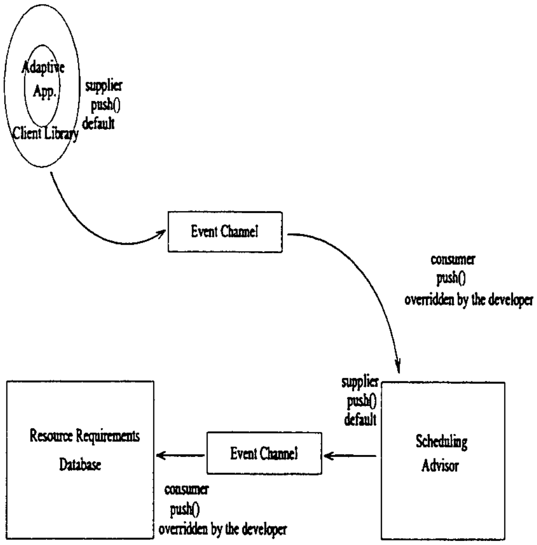


Figure 11: Using **push()** Operation

3.3 Remote Invocations

In this section, we discuss using remote invocations to coordinate the interactions of MSHN's components. Since both the Static Invocation Interface (SII) and the Dynamic Invocation Interface (DII) have similar remote invocation mechanisms, we first define the general problems encountered with both, and then enumerate any additional ones that are specific to the DII.

The same functionality described above using the Event Service can be implemented using remote invocation. The most important difference is that the replication of the components is not as easy as it is using Event Service. To support replication using remote invocation, clients must make multiple invocations rather than just the one needed in Event Service.

3.3.1 General Approach using Remote Invocation

Figure 12 shows our approach that uses remote invocations (i.e., either the Static Invocation Interface (SII) or the Dynamic Invocation Interface (DII)) to establish inter-component communication in the MSHN architecture. We chose from two communication methods available in both the SII and DII: one-way invocation and synchronous invocation, depending upon whether reliable communication is required.

When using the SII, a component requires compile-time knowledge of the Interface Description Language (IDL) interface of the target component from which it will request a service. In contrast, the same component, using the Event Service, makes its request via a

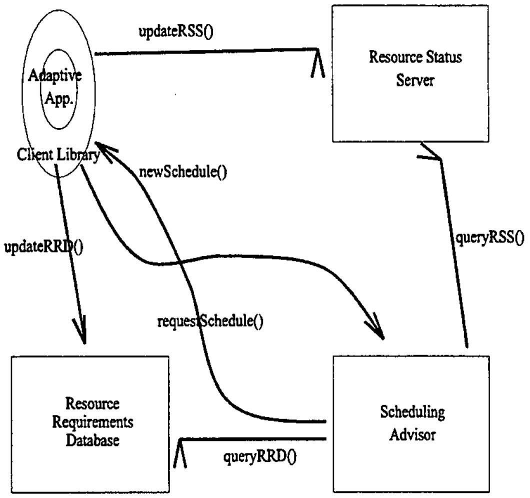


Figure 12: Using Remote Invocations in MSHN

standard API that is independent of the target component and its functionality. However, when using the DII, the components of MSHN can invoke operations on other components without requiring precompiled stubs. Thus, we may substitute different instantiations of such components without requiring a re-linking. Additionally, using the DII allows us to invoke objects using deferred synchronous invocation. Such invocation is not available from the SII within the current CORBA 2.2 specification. With deferred synchronous invocation, the clients may continue their computation instead of waiting for the results of the previously invoked operations to be delivered.

3.3.2 Problems with Using the Initial Remote Invocation Approach

We now enumerate some problems with our initial remote invocation approach.

Lack of a Standard Thread Mechanism. Our first design decision was to implement the remote invocations with threads, i.e., handling each invocation of a component using a different thread. Using threads would avoid any data synchronization problems and support fairness for each schedule request. However, the CORBA 2.2 specification does not define how the threads must be implemented. Therefore, each vendor has come up with their own solution, leading to applications that are non-portable. For example, if you use IONA's Orbix as your development environment, and IONA's Filters to implement your threads, you cannot use the same implementation on Inprise's Visibroker because Inprise's solution for handling threads uses Interceptors.

We avoided non-compliant extensions of the vendor when implementing our prototypes. Therefore, we were unable to use threads for any of our prototypes, although the usage of threads would have improved the throughput of schedule requests.

Best-Effort Semantics. One-way invocation has best-effort semantics. Thus, there is no guarantee that the requested method is actually invoked. In this mechanism, the client continues its processing immediately after initializing the request and never synchronizes with the completion of the request. Hence, one-way invocation is not a good mechanism for most of the MSHN system because it is not reliable.

However, using one-way invocations for frequent short-term updates could be cost effective in some cases in MSHN. There are two advantages to selectively using best-effort asynchronous semantics between MSHN's Client Library and Resource Status Server. First, the Client Library can continue its computation immediately without blocking. Second, we expect that the Resource Status Server will be updated very frequently. Therefore, we can afford the delay needed to get the accurate status of a resource with the next update instead of forcing the use of a more reliable transmission mechanism.

3.3.3 Problems with Our Initial Approach that are Specific to using DII

We now enumerate some problems with our initial approach that are specific to using DII.

The Additional Overhead of the DII. A straight forward DII approach requires 5-6 method invocations in order to invoke a single remote method: looking up the interface name, getting the operation identifier/parameters, and creating the request (which may also be remote). This would add a lot of overhead to run-time performance, which would be unacceptable in MSHN's architecture.

In MSHN however, we know the interface of the components, i.e., the operation identifier, the parameters and the return type, when we are developing the client applications. Thus, we can obtain the flexibility and benefits the DII's deferred synchronous invocation, without having to pay the overhead of querying the Interface Repository for the interface information. We do note that if a deferred synchronous invocation, such as Promises [16], had been specified as part of CORBA's static invocation interface, the use of DII would not be necessary in this case. We compare the performance of the SII and DII in the results section.

3.4 Using the Naming Service

We used the Naming Service to obtain object references in each of our prototypes. For the static and dynamic invocation interfaces, all components must resolve names only once, when they are instantiated, to obtain IORs via the Naming Service. References within all components, except the Client Library, are stored in files for future use as we described previously. The components do not use the Naming Service unless the IORs that they have are no longer valid. We use the exception handling mechanism in CORBA to catch non-valid IORs, and then use the Naming Service to obtain new valid ones.

To improve the run-time performance of the Event Service implementations, we registered each component with the appropriate Event Channel. We resolve the Event Channel references using the Naming Service. Then we query the Event Channels to obtain the references for the Proxy Push Suppliers, stringify them, and then store them in files. When a component receives an event, and generates another event in response to the one it received, that component reads the appropriate file to obtain the stringified reference and uses this reference to push the event to the corresponding Event Channel.

4 Quantitative Results

We described our design decisions for implementing our prototypes in the previous section. In this section, we discuss the performance results of these different prototypes. First, we describe our test bed. Then we explain our tests and enumerate their results.

4.1 Hardware and Software Used in the Test Bed

As discussed earlier at the beginning of this research, we surveyed the available implementations of CORBA to determine what services were supported. (See Figure 7 and 8.) Based upon the robustness and availability of services, particularly the Typed Event Service, we chose IONA Technologies' CORBA implementation, specifically OrbixMT2.3c, OrbixNames1.1c, OrbixEvent1.0c (Untyped Event Service) and OrbixEvent1.0b (Typed Event Service) built using the SunSparc C++ Compiler 4.1.

We ran our tests on SunSparc Station 10 hosts with 300MHz CPUs and 128 MB of RAM each, running the Solaris 2.6 operating system. The hosts were connected via a 100 Mbits/sec Ethernet LAN.

To obtain correct results in the tests utilizing the network, we used the Network Time Protocol to synchronize the system clocks of the hosts. We found that the system clock on the SunSparc 10 has a skew of approximately 3 milliseconds every 15 minutes. Therefore in order to minimize the difference between the various system clocks, we synchronized the clocks every 5 minutes and ran the tests immediately after the synchronization.

4.2 Experiments

We determined the overhead of each CORBA mechanism on a single machine, and then measured the response times over the network of the various mechanisms, that is, the total time required to service 1000 scheduling requests. This interval begins when the Client Library requests a schedule from the Scheduling Advisor and includes all processing up until the time that the Client Library receives a response. This duration includes the time spent querying the Resource Requirement Database and the Resource Status Server. At the time of this testing, we did not have a fully functional Scheduling Advisor, so we emulated its execution by having the thread that was computing a schedule pause for .5 seconds. We chose this duration based upon the average execution time of a set of 11 scheduling algorithms proposed for MSHN's repertoire by Siegel [17].

To assess the overhead of CORBA, we included one non-CORBA test. This base case consists of an application linked with all the MSHN components and executing as a single process on a single host. This non-CORBA test uses local method invocation to perform MSHN component intercommunication. In order to assess CORBA's overhead, we performed two sets of tests. In the first set, we compared this base case against test cases where we ran all the MSHN components on the same machine and had them communicate via CORBA mechanisms. In the second, we compared the latter tests against ones where the MSHN components are distributed across different machines.

With the exception of the non-CORBA base case, we ran all tests both on a single machine and over the network using different workstations to execute each of the Client Library, the Resource Status Server, the Resource Requirements Database and the Scheduling Advisor.

All single machine CORBA tests were executed using four different processes. The non-CORBA single machine tests executed completely in a single process, with all MSHN calls being implemented as ordinary C++ function calls. In implementing both static invocation and dynamic invocation for a single machine, we used synchronous semantics.

The average inter-arrival rate of schedule requests varies with the facility and time of day. Therefore, we ran all of our tests for two different circumstances. In the first, the inter-arrival rate of the requests is less than the service time, i.e., each request is completed by the system before the next request arrives on average. The second represents the situation that exists in the middle of a burst. In this case, the inter-arrival rate of the requests is greater than the service time, i.e., some requests must be queued to be handled later. The first case is important in determining performance under normal conditions, but it is equally important for us to determine that the system neither (1) fails completely when heavily loaded, nor (2) incurs overhead that varies exponentially with the number of requests pending. Indeed, no typed event service that we have tested to date could pass the above stress tests.

Unfortunately, the system clocks had insufficient granularity to measure precisely the total time to process a single request in our non-CORBA implementation. We therefore first read the system clock. We then generate a request and await its response, repeating this 1000 times. Lastly, we read the clock again, and determine the total time (for 1000 consecutive request-response pairs). Because requests are generated consecutively, and because each request uses synchronous semantics to make the invocations, we call this set of tests, the **consecutive synchronous tests**.

To simulate the case where many requests occur within a short time frame, we generated requests every .06 seconds, on average, in our base case. For this set of tests, we used asynchronous calls within the application to start the schedule request chain in the DII and SII implementations. Event Service is meant to be used asynchronously, so there was no special programming required to implement these cases. We call this set the **bursty asynchronous** tests because during such a burst, the requests arrive faster than the expected required service time and queue up for the Scheduling Advisor.

For another of our projects, Schnaidt and Duman implemented a fully optimized version of an application using sockets and compared it to an equivalent CORBA implementation to determine CORBA's overheads when running over the network [18]. As such, we did not implement such a socket implementation of MSHN. In the following paragraphs, we draw some conclusions based both on the Schnaidt-Duman experiments and those reported here.

4.3 Results

We summarize our quantitative results in Figure 13. The times shown are the actual execution times, in seconds, for 1000 requests. We have included a scheduling time of .5 seconds per request and have not simulated the execution time of the application.

In order to fully understand these results, we must first explain some anomalies that we observed in the Unix calls we used to emulate the Scheduling Advisor (`select()`) and the request generation inter-arrival rate (`ualarm()`). The average of the actual `select()` times was 125 microseconds more than the requested .5 seconds. We also observed an average of 10 milliseconds error for the `ualarm()` requests of 60 milliseconds.

Config.	**Communication Mechanism**	**Local**	**Network**
Consec. Synch.	Non-CORBA	500.1	N/A
	SII	511.4	520.0
	DII	530.1	530.4
	Untyped Event	607.4	593.9
	Typed Event	580.5	779.2
Bursty Asynch.	Non-CORBA	500.1	N/A
	SII	510.8	510.8
	DII	521.2	520.2
	Untyped Event	592.8	564.4
	Typed Event (for 100 requests)	64.7	63.6

Figure 13: Results of the Generic Experiments for 1000 Requests

As expected, there is significant overhead in using CORBA for communication, and therefore across more than one address space, as compared to local invocations within a single address space. In our earlier project, we noted similar results as well as substantial overhead when an optimized non-CORBA local socket implementation was compared to a local CORBA implementation [18]. The efficiency of the socket implementation on a single machine is due to its use of shared memory. However, even if a CORBA implementation used shared memory, comparable performance would not be obtained. Unfortunately, the CORBA specification requires all parameters of a request to be converted to an external, machine independent data representation, even if the target object resides on the same machine. Also, in that earlier project, we noted that a networked CORBA implementation, which required less than 5% of the time to implement as compared to the socket implementation, had only 20% more runtime overhead. Since our results are comparable here, and because we did not implement a highly optimized MSHN socket implementation, we will limit the remainder of our remarks to comparing the performance of various CORBA implementations of MSHN.

Static invocation is generally the fastest intercommunication mechanism available in CORBA [1]. Even though dynamic invocation is generally much slower, we see that the performance of dynamic invocation, when we know the interfaces at development time, is close

Communication Mechanism	Added Overhead
SII	10.5
DII	20.0
Untyped Event	64.2
Typed Event	13.5

Figure 14: Added Overhead for Bursty Asynchronous Test Case over the Network

to that of static invocation. However, we note that the most efficient implementation would likely be available from a deferred synchronous Static Invocation Interface. We recommend that such semantics, similar to those in Promises [16], be considered for adoption into the CORBA specification.

The comparison between the consecutive synchronous and bursty asynchronous tests seems surprising at first glance. One would normally expect that a system loaded with bursty requests would not perform better than an unloaded system. To understand the reason for this performance improvement, we must further elaborate on the client application's use of the Naming Service. In the consecutive case, the Client Library obtains the reference of the Scheduling Advisor from the Naming Service immediately prior to making each request. However, in the bursty asynchronous case, the Client Library obtains all of the references asynchronously. Thus in the bursty asynchronous case, obtaining these references overlaps with the actual computation. Unfortunately, we will only expect to see this improvement in the actual MSHN implementation if the Scheduling Advisor is executing on a dual processor machine. In our experiments, the emulated Scheduling Advisor is actually blocked while the Naming Service is resolving addresses.

In the 4-machine network tests, the number of context switches required between MSHN's components and the Object Request Broker is substantially reduced. Multiple components actually execute simultaneously, and thus run-times were smaller.

As seen in Figure 13, the Untyped Event Service adds more overhead than either static or dynamic invocation because the Event Service process is the bottleneck in the system. Of course in an overall evaluation, this additional overhead must be balanced against the reduced cost with which information can be delivered to replicated system components.

In addition to the tests described above, we replicated the Untyped Event Service to see whether any speedup could be obtained by distributing the load of the Event Service process. First we created two Event Service processes, one on the same host as the application and the other on the same host as the Scheduling Advisor, in an attempt to achieve some speed up. This approach performed worse than the single Event Service process. Upon analysis, we determined that it introduced unnecessary network communication and placed the Event Service processes on the busiest hosts. Then we moved the Event Service processes to the same hosts as the Resource Requirements Database and the Resource Status Server. Figure 15 shows the speedup we observed with this configuration. We also ran tests using four distributed Event Service processes. Unfortunately, probably because of the excessive amount of communication, this approach performed no better than using a single Event Service process.

In MSHN's Typed Event Service implementation, all of the communication passes through a single process. The CORBA implementations that we used[7] failed in this bursty asynchronous case. In Figure 13, we include the time required to process 100 requests for the bursty asynchronous case. Since the current implementations of Typed Event Service do not allow replication, we could not run a replicated test with the Typed Event Service as we did with the Untyped Event Service. Hence, we believe that the Typed Event Service is not ready to be used in middleware to support heterogeneous distributed computing.

5 Conclusions

In this paper, we described our experiences using mechanisms of the CORBA 2.2 specification to facilitate communication in a resource management system that is both designed to manage distributed heterogeneous applications, and is itself distributed and heterogeneous. In our qualitative assessment of CORBA 2.2, we found several minor problems and recommended the addition of deferred asynchronous semantics to CORBA's Static Invocation Interface. We found that both CORBA's static invocation and dynamic invocation, when used solely to obtain asynchronous semantics, were efficient enough to support distributed heterogeneous resource management systems. We found that substantial work is needed to provide implementations of Typed Event Services that can handle the loads placed on them when requests occur in a bursty fashion. We also determined that while Untyped Event Services add substantial overhead as compared to static invocation, they may still be desirable in the case where multicast of requests is desired, particularly if they are replicated and themselves wisely allocated to machines in the system. In summary, many of the

[7] Typed Event Service is new in the CORBA 2.2 specification and not many CORBA products have this service available as yet.

	Replication Mechanism	All	SA and Client Hosts	RRD and RSS Hosts
Bursty Asynch.	Two Event Pro.	N/A	574.38	561.44
	Four Event Pro.	560.98	N/A	N/A
Consec. Synch.	Two Event Pro.	N/A	599.05	593.82
	Four Event Pro.	593.82	N/A	N/A

Figure 15: Results of the Untyped Event Service Special Cases

existing CORBA services can be quite useful in implementing resource management systems for heterogeneous computing, and other CORBA services hold substantial promise for the future.

6 Acknowledgements

The authors would like to thank Ted Lewis for his suggestions during this research as well as for sharing his broad understanding of the motivation behind the CORBA specifications.

References

[1] Robert Orfali, Dan Harkey, and Jeri Edwards. *Instant CORBA*. John Wiley, New York, 1997.

[2] Robert Orfali, Dan Harkey, and Jeri Edwards. *Distributed Objects*. John Wiley, New York, 1997.

[3] Sean Baker. *CORBA Distributed Objects Using Orbix*. Addison Wesley Longman Limited, Essex, 1997.

[4] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Client/Server Survival Guide*. John Wiley, New York, 1997.

[5] IONA Technologies PLC. *Orbix Programmer's Reference Manual*, October 1997.

[6] IONA Technologies PLC. *Orbix Programmer's Guide*, October 1997.

[7] Object Management Group. *CORBA 2.2 Specification*, February 1998.

[8] Object Management Group. *Naming Service Specification*, November 1997.

[9] Object Management Group. *Event Service Specification*, November 1997.

[10] Steve Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), February 1997.

[11] Douglas C. Schmidt and Steve Vinoski. Overcoming drawbacks in the OMG events service (column 10). *SIGS C++ Report Magazine*, June 1997.

[12] Debra A. Hensgen, Taylor Kidd, Matthew C. Schnaidt, David St. John, Howard Jay Siegel, Tracy D. Braun, Muthucumaru Maheswaran, Shoukat Ali, Jong-Kook Kim, Cynthia Irvine, Tim Levin, Roger Wright, Richard F. Freund, Michael Godfrey, Alpay Duman, Paul Carff, Shirley Kidd, Viktor Prasanna, Prashanth Bhat, and Ammar Alhussaini. An overview of MSHN: A Management System for Heterogeneous Networks. In *8th. IEEE Workshop on Heterogeneous Computing Systems (HCW'99)*, San Juan, Puerto Rico, April 1999. IEEE, IEEE. invited.

[13] John Kresho. Quality network load information improves performance of adaptive applications. Master's thesis, Naval Postgraduate School, September 1997.

[14] IONA Technologies PLC. *OrbixEvents Programmer's Guide*, December 1997.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[16] B. Liskov and L Shirira. Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. *Proceedings SIGPLAN'88 Conference Programming Design and Implementation*, 1988.

[17] Tracy D. Braun, Muthucumaru Maheswaran, Howard Jay Siegel, Noah Beck, Ladislau Boloni, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, and Bin Yao. A taxomony for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. *Proceedings of the Workshop on Advances in Parallel and Distributed Systems (ADAPS)*, 1998.

[18] Matthew Schnaidt and Alpay Duman. A comparison of Unix sockets and CORBA in a distributed communication intensive application. Technical Report 3, Naval Postgraduate School, 1998.

Alpay Duman is LTJG in Turkish Navy. He graduated from the Turkish Naval Academy getting his BS in Operations Research with honor degree. He received his Ms.Cs. degree in the area of Systems Design and Architecture from the Naval Postgraduate School. He investigated the use and runtime overhead of CORBA in DARPA-sponsored Management System for Heterogeneous Networks QUORUM project (MSHN) with Dr. Debra Hensgen and Dr. Ted Lewis. He is currently a systems engineer at Turkish Navy Software Development Center working on a CORBA based communication infrastructure for Command Control Systems. His area of interest are fault tolerant, real-time, CORBA based distributed systems.

Debra Hensgen is an Associate Professor in the Computer Science Department at The Naval Postgraduate School. She received her PhD in the area of Distributed Operating Systems from the University of Kentucky. She is currently a Principal Investigator of the DARPA-sponsored Management System for Heterogeneous Networks QUORUM project (MSHN) and a co-investigator of the DARPA-sponsored Server and Active Agent Management (SAAM) Next Generation Internet project. Her areas of interest include active modeling in resource management systems, network rerouting to preserve quality of service guarantees, visualization tools for performance debugging of parallel and distributed systems, and methods for aggregating sensor information. She has published numerous papers concerning her contributions to the Concurra toolkit for automatically generating safe, efficient concurrent code, the Graze parallel processing performance debugger, the SAAM path information base, and the SmartNet and MSHN Resource Management Systems

David St. John is the head of staff at the Heterogeneous Network and Computing Laboratory. He plays a strong role in development of the MSHN prototype and in directing students' research at the Naval Postgraduate School. He has over six years experience in object-oriented software development primarily for process control, sensor collection, and Internet transaction processing systems. He is a member of IEEE and IEEE Computer Society. He was a recipient of the Chancellor's Fellowship and an MS degree in Engineering from the University of California, Irvine in 1994. He received his BS degree in Mechanical Engineering from the University of Florida in 1992.

Taylor Kidd obtained his Ph.D. from the University of California at San Diego in 1991. He is an active researcher in theoretical and applied distributed computing. As part of the SmartNet Team at NRaD he led the SmartNet Research Team. Recently he joined Debra Hensgen as co-director of the Heterogeneous Network and Computing Laboratory. While part of the SmartNet Team, he instigated a number of important advances to the SmartNet Scheduling Framework, including enhancing compute characteristic learning by using Student-T techniques, performing experiments and simulations exploring the performance of different RMS's in homogeneous and heterogeneous environments, and fundamentally changing the way SmartNet learns and schedules by recognizing the basic predictable uncertainty of job run times. He has served on the program committees of several conferences and has worked as an acting subject area editor for JPDC. Most recently, he is working under DARPA's Quorum program as a co-PI for MSHN and as a co- investigator on the SAAM Project.

An Overview of MSHN: The Management System for Heterogeneous Networks

Debra A. Hensgen[†], Taylor Kidd[†], David St. John[§], Matthew C. Schnaidt[†], Howard Jay Siegel[‡], Tracy D. Braun[‡], Muthucumaru Maheswaran[¥], Shoukat Ali[‡], Jong-Kook Kim[‡], Cynthia Irvine[†], Tim Levin[§], Richard F. Freund[µ], Matt Kussow[µ], Michael Godfrey[µ], Alpay Duman[†], Paul Carff[†], Shirley Kidd[§], Viktor Prasanna[¶], Prashanth Bhat[¶], and Ammar Alhusaini[¶]

[†]Department of Computer Science
Naval Postgraduate School
Monterey, CA USA

[‡]School of Electrical and
Computer Engineering
Purdue University
West Lafayette, IN USA

[¶]Electrical and Computer Engineering
University of Southern California
Los Angeles, CA USA

[µ]NOEMIX
San Diego, CA USA

[¥]Department of Computer Science
University of Manitoba
Winnipeg, Canada

[§]Anteon Corporation
Monterey, CA USA

Abstract

The Management System for Heterogeneous Networks (MSHN) is a resource management system for use in heterogeneous environments. This paper describes the goals of MSHN, its architecture, and both completed and ongoing research experiments. MSHN's main goal is to determine the best way to support the execution of many different applications, each with its own quality of service (QoS) requirements, in a distributed, heterogeneous environment. MSHN's architecture consists of seven distributed, potentially replicated components that communicate with one another using CORBA (Common Object Request Broker Architecture). MSHN's experimental investigations include: (1) the accurate, transparent determination of the end-to-end status of resources; (2) the identification of optimization criteria and how non-determinism and the granularity of models affect the performance of various scheduling heuristics that optimize those criteria; (3) the determination of how security should be incorporated between components as well as how to account for security as a QoS attribute; and (4) the identification of problems inherent in application and system characterization.

1. Introduction

The Management System for Heterogeneous Networks (MSHN[1]) project seeks to determine an effective design for a resource management system (RMS) that can deliver, whenever possible, the required quality of service (QoS) to individual processes that are contending for the same set of distributed, heterogeneous resources. Factors influencing QoS requirements include security, user preferences for different versions of an application, and deadlines. A set of QoS requirements, considered together with resource availability, determine whether all processes' requirements can be met.

An RMS, also sometimes called a meta-computing system, is similar to a distributed operating system in that it views the set of machines that it manages as a single virtual machine [51]. Also, like any distributed operating system, it attempts to give the user a location-transparent view of the virtual machine. Hence, as in the case of a distributed operating system, an RMS provides users with improved performance while the location of resources is hidden. The set of users of a system, which consists of both local and remote resources, that is managed by an RMS should be able to attain a higher level of availability and more fault tolerance than would be available from their local system alone.

This research was supported, in part, by the DARPA/ITO Quorum Program.

[1] Pronounced, "mission"

An RMS differs from a distributed operating system in that it does not micro-manage the resources of each computer. Instead, each computer runs its native operating system. Similarly, each router executes its own protocol and each file server executes a native distributed file system. The RMS is responsible for identifying the large-grained resources, i.e., compute servers and data repositories that should be used by each process, if there is a choice. It may be responsible for issuing a command to begin execution of the processes that comprise an application. It may monitor the status of both the resources in the system and the progress of the applications for which it is responsible.

It is unclear whether every request to execute an application that is submitted to any operating system on any of the machines in the distributed system must be controlled by the RMS. If all requests are controlled by the RMS, then allocation policies that attempt to optimize throughput for a set of well-understood applications will perform better. However, sometimes users wish to maintain control over which resources their application will use.

There are many active, on-going research projects, in addition to MSHN, in the area of resource management, and there are many major research problems to be solved. A problem that MSHN is not addressing is the best way for such a system to interact with human users to obtain their QoS preferences and requirements in the most user-friendly way. Indeed, simply identifying the syntax and semantics required to express all of the QoS preferences and requirements is a difficult problem [13][17][37][55]. While MSHN does not address this problem, the designers of MSHN expect to leverage results from research in this area. They assume, for example, that a request to execute an application is accompanied by a list of deadlines, preferences for various versions of an application, security requirements, and any restrictions on the variance of the time at which a request should be completed.

Before leaving the general topic of RMSs, it is imperative that we address the topic of "packaging." MSHN researchers do not see the fruits of the RMS research as a large, monolithic piece of software that will require its own separate installation and maintenance. The best way to package the eventual outcomes of the RMS projects may be to incorporate them into an infrastructure- or middleware-level standard similar to the Common Object Request Broker Architecture (CORBA), Domain Name Services, or other such resource location services. In this way, an RMS would not need to be separately maintained and would be consolidated with the services that distributed applications will most often use. However, it is still worthwhile to separate research on RMSs from research in all other aspects of distributed object computation that will be needed in future versions of such standards in order to first isolate, then solve some of the difficult resource management problems.

1.1. Background

MSHN evolved in part from a scheduling framework called SmartNet [19][29]. SmartNet's goal was to be able to wisely schedule sets of compute-intensive jobs, some of which may require the execution of multiple processes, onto members of a suite of heterogeneous computers. SmartNet provides a sophisticated scheduling module that had been successfully integrated with many RMSs and distributed computing environments. Hence, users who need to execute compute-intensive jobs and have access to a shared, heterogeneous environment can achieve superior performance, while continuing to work in an environment to which they have grown accustomed [23]. Additionally, for those users who do not already have one installed, SmartNet provided a basic RMS that makes use of its sophisticated scheduling capabilities. SmartNet's major research contributions include:

- The ability to predict the expected run-time of a job on a machine using the concept of compute characteristics and information collected from previous executions of the job.
- The ability to leverage the heterogeneity inherent in both a collection of jobs as well as in a collection of computers.

SmartNet was used successfully by DoD and the National Institutes of Health in scheduling their compute-intensive jobs, and by NASA's EOSDIS system in determining whether their resources were adequate to process data in the ways desired by their scientists.

SmartNet's scheduling algorithms are tuned to attempt to minimize the time at which the last job completes, although the designers of SmartNet recognized that similar algorithms may be useful in optimizing other criteria. Of course, minimizing the time at which the last job, of a set of jobs, completes is, in general, an NP-complete problem, so SmartNet employs heuristics when it searches for a near-optimal mapping of jobs to machines and job execution schedule. Many of the heuristics that it uses are well known and previously documented, however, they had not previously been used in a practical heterogeneous computing system [25]. It is likely that they were not previously used in actual systems because system designers had not tried to estimate average process run-times and because it was not previously recognized that exact run-times, though helpful, were not necessary [2][3][28].

1.2. Overview of MSHN's goals

MSHN differs from SmartNet in three major ways. First, SmartNet was expected, from the beginning, to be a system that would actually be used in production. For this reason, much of the SmartNet developers' time was spent ensuring that SmartNet was at SEI Level 3. Despite this, SmartNet was able to make significant research contributions. MSHN is intended to be a research system, facilitating experiments by the investigators to determine how RMSs, that have somewhat broader goals than SmartNet, can be built. MSHN's research goals expanded upon SmartNet's in the following areas.

(i) MSHN needs to consider that the overhead of jobs sharing resources, such as networks and file servers, can have significant impact on mapping and scheduling decisions.

(ii) MSHN must support adaptive applications (defined below).

(iii) MSHN must deliver good QoS to many different sets of simultaneous users, some of whom may be executing interactive jobs; others, compute-intensive jobs; and still others, real-time requirements.

In SmartNet's model, applications consist of three distinct phases. In the first phase, which is short compared to the second phase, they acquire data from a data repository. In the second phase, they compute results based upon the data that they obtained during the first phase. In the third phase, which is again very short compared to the second phase, they write the result back to a possibly different repository. Because the first and third phases are so short, SmartNet's heuristics assume that there is no contention for either the network or the data repositories. However, they do account for the time required to access the resources, assuming that each application is the sole user of those resources. The model of applications that MSHN is meant to manage is more complex, permitting applications to transition through many more phases of variable length, each requiring not only sharing of compute resources, but also sharing of network and data repository resources. We discuss briefly in this paper, and elaborate elsewhere, both the problem of modeling the application and that of accounting for lower level policies that govern the sharing of resources. That is, because MSHN does not assume that it has any control over network routing, file server memory allocation, etc., it models, when necessary, the lower level operating systems and protocols. By doing so, the assignment of processes to resources will account for the sharing of those resources in the correct way.

The second major difference between SmartNet and MSHN's research goals is that MSHN attempts to provide support for **adaptive** and **adaptation-aware** applications. By adaptive applications, we mean idempotent applications that can exist in several different versions. Different versions may have different values to a user due to factors such as precision of computation or input data. Additionally, different versions may have different communication and computation needs. Or, one version may execute on Windows NT while another version is an executable for Linux. MSHN's goal is to support adaptive applications by being able to terminate one version of an application if MSHN perceives that the currently executing version will not meet the users' QoS expectations.[2] In that case, MSHN would terminate the executing version and start up another version from the beginning (if there were sufficient resources to execute that other version). The requirement that adaptive applications be idempotent permits the application to be safely restarted from the beginning without corrupting any resource such as a database. Similarly, there may be times when MSHN determines that delivery of a better QoS is possible to a user by changing to a version that better meets that user's preferences.

An adaptation-aware application differs from an adaptive application in two ways. First, when it is terminated, the new version need not be restarted from the beginning. Instead, a different version from the one that terminated may be started, using information about a previous state that was obtained from the execution of the previous version. Second, an adaptation-aware application may be able to adapt its resource usage during execution, without restarting.

Finally, MSHN's goals differ from SmartNet's in that MSHN seeks to determine how to meet multiple different QoS requirements to multiple different applications simultaneously. There are really two issues bound up in this difference. First, a way to incorporate, dynamically, the mixture of QoS requirements into a single measure must be determined. Second, an assignment of applications to resources must also be determined that optimizes the identified measure. In resolving this second issue, we can strongly leverage SmartNet's emphasis on the separation of optimization criteria and search algorithms and the recognition that similar algorithms can be used to search many different types of spaces for optimal values. We elaborate on this below.

1.3. Related work

There are other research groups examining the issues important to building an RMS, many within DARPA's Quorum project. Here, we look at some of the projects related to MSHN. Some of these groups are engaged in research complementary to MSHN's goals. For the sake of brevity, only a short synopsis of each project, as it relates to MSHN, is presented.

DeSiDeRaTa. The University of Texas at Arlington has a project called "DeSiDeRaTa: QoS Management Tools for Dynamic, Scalable, Dependable, Real-Time

[2] We note that a version of one application may be terminated because MSHN detects that another user's application will not meet its QoS expectations. This phenomenon can occur due to priorities.

Systems." DeSiDeRaTa is focusing on QoS specification, QoS metrics, dynamic QoS management, and benchmarking of specific computing environments, such as the distributed Anti-Air-Warfare system at the Naval Surface Warfare Center, Dahgren Division. A unique concept that has come out of the DeSiDeRaTa project is that of an application "path" [56].

Globus. Globus is a large, joint project from Argonne National Laboratory and the University of Southern California's Information Sciences Institute. Parts of the Globus project are devoted toward resource management issues. The Globus architecture depends on an advance or immediate resource reservation protocol layer, for which a standard does not yet exist [14][18].

RT-ARM. Honeywell is developing a "Real-Time Adaptive Resource Management" system aimed primarily at high-end, real-time military embedded systems such as the Navy Surface Combatant Ship SC-21. Some of the specific issues they are concentrating on include modeling embedded systems and finding practical techniques for predictable real-time performance [24].

EPIQ. The EPIQ project, from the University of Illinois at Urbana-Champaign, is building an infrastructure for providing guaranteed QoS features, upon which RMSs may be built. part of their infrastructure involves building their own runtime environment [35].

ERDoS. SRI International is running a project called ERDoS (End to End Resource Management for Distributed Systems) which is developing an architecture for adaptive QoS-driven resource management. The ERDoS project emphasizes a comprehensive definition of QoS and the development of models that capture information required for making resource management decisions [46].

QUASAR. The QUASAR (QUAlity Specification and Adaptive Resource management for distributed systems) project, at the Oregon Graduate Institute of Science and Technology, is investigating techniques for specifying and utilizing QoS in adaptive, distributed systems. QUASAR is concentrating on the translation of QoS specifications from the application-level to the resource-management-level, and its use in reservation-based resource management, primarily in the multimedia domain [53].

ASSERT. The ASSERT System at the University of Oregon, Eugene, is focusing on dynamic, distributed, real-time environments. The core of the project estimates and monitors the relevant QoS parameters of running applications. ASSERT is not an RMS, nor an RMS framework; rather, the ASSERT project is looking at a specific issue of RMSs: QoS monitoring and estimation [16].

QuO. The Quality Objects (QuO) project, from BBN Systems Technologies, is attempting to add QoS specification and delivery to CORBA. Rather than provide absolute QoS guarantees, QuO seeks to combine knowledge about resource and application conditions in order to reserve enough end-to-end resources for predictable execution of distributed applications [52].

MOL. The MOL (Metacomputing OnLine) project from the Paderborn Center for Parallel Computing has as a goal the utilization of multiple high performance systems for solving problems too large for a single supercomputer. The MOL approach does not assume absolute control of resources under its management. The MOL project is addressing several of the issues key to resource management, including QoS specification [42].

1.4. Organization of the paper

In the next section of the paper we motivate and discuss MSHN's architecture. Even though SmartNet was successful in achieving its functionality, rather than using SmartNet's architecture exactly, we based MSHN's architecture upon lessons learned from SmartNet, because MSHN's goals are substantially different. In particular, we clearly delineated certain of SmartNet's modules into separate components. This delineation makes it easier to experiment with different designs for each of the components. In section 3, we then discuss many of the research issues that the MSHN investigators are studying and highlight some of the results. Additionally, this section provides references to the numerous articles that describe this research in more detail. We conclude by summarizing the status of the MSHN project.

2. MSHN's architecture

In this section, we first describe some of the concepts that went into MSHN's architectural design. This description motivates the need for the various major components and explains why they must be replicated to varying degrees. The architectural design was driven by the need to support the RMS research that we will discuss in the next section and was aided by our previous experience with SmartNet. We then present MSHN's current architecture in detail.

2.1. Motivation

We first motivate the need for each of the major components of MSHN's architecture, then discuss how those components interact with one another.

We recall from the previous section that an RMS needs to transparently locate the resources that should be used when execution of an application is requested. Therefore, it must be made aware of any request, by either a user or an application, to start executing another application. Many early RMSs required the user to explicitly log in to the system to start a job. If an application was to be started from within another application, e.g., through fork and exec system calls, then the application that makes the request would be required to be specially designed to embed these requests within a function call to an RMS library. This restriction required that applications be specifically written or modified for a particular RMS.

The MSHN designers do not want to force a user to explicitly log into an RMS, or to modify their existing programs. Instead, MSHN transparently intercepts calls to system libraries that would otherwise initiate execution of a new process and diverts those calls to a MSHN Client Library. After MSHN decides where the newly requested application should execute, the MSHN Client Library uses whatever mechanisms available at the resource site to initiate execution of the remote process.

The environments for which MSHN is designed contain many different types of computers, each possibly executing a different version of an operating system. Rather than requiring the Client Library, which is linked with every MSHN application, to contain a substantial amount of code that is specific to each of these computers, we chose to make use of a MSHN Daemon. Whenever a computer is added to a system, a MSHN Daemon is started on that computer. When a Client Library needs to start a process on a remote machine, it simply contacts the MSHN Daemon on that machine and requests that the Daemon start the process on the Client Library's behalf. Of course, the general mechanism that we use in the Daemon is not new, and is therefore not a research issue.

When a remote process needs to communicate with the initiating process, it contacts the Client Library, which passes the information on to the initiating process, just as though the remote process were started locally. Being able to transparently provide this service to applications, whether or not they are command interpreters, requires that the Client Library intercept, and at least pre-process if not divert, other system library calls in addition to the previously mentioned exec call. For example, all of the socket calls and all calls to open, close, read, and write files must be intercepted and replaced or at least pre- and post-processed.

The MSHN project required a mechanism for intercepting these calls without requiring source modification. We initially turned to the Condor project for help with this problem [36]. Condor is a project at the University of Wisconsin that performs transparent migration of processes in a Unix environment. To perform this migration, Condor also had to intercept these calls to system libraries. Using techniques similar to those used by Condor, we were able to intercept these calls without requiring source code modification.[3] The mechanism is described in detail elsewhere [44].

In addition to providing a mechanism for transparently executing remote processes, the Client Library is in a unique position to passively determine the status of resources, because it is assumed to be linked with any application executing in an environment managed by MSHN. That is, the MSHN Client Library can pre- and post-process system calls, because it is intercepting all such calls made to the operating system, which are executed when a process needs to use a hardware resource. In so doing, it can determine the low level, end-to-end QoS that an application is receiving from a particular resource. We will discuss this functionality of the Client Library further in the next section.

When the MSHN Client Library intercepts a call to execute a new process, it must have some way of determining which resources that new process should use, i.e., which computer should primarily be responsible for executing the new process.[4] Rather than requiring that decision to be made independently by each Client Library that is linked with each application, we chose to have the Client Library first check the request against a list of applications managed by MSHN. If the requested application is not on that list, the MSHN Client Library simply passes the requested application directly to the local operating system. If the requested application is on that list, it instead passes the request to the MSHN Scheduling Advisor. It is the Scheduling Advisor's job to determine which set of resources the newly requested process should use.

The MSHN Scheduling Advisor is itself a complex package, associated with many different research issues which we discuss more fully in the next section. Among the primary research issues are: (i) what criteria should be optimized in the choice of resources? (ii) Because optimizing the criteria is likely to be an NP-complete problem, if n is too large, which heuristic should be used to search for an optimum resource assignment? (iii) With what granularity must the Scheduling Advisor model both the policies and protocols associated with allocation of the lower level resources and what granularity of model should it use to define the resource requirements of a process?

For the Scheduling Advisor to determine a good assignment of resources for a process, it must know both which resources and how much of each resource would be required for a process to execute and meet its QoS requirements and preferences. Therefore, to assist the Scheduling Advisor in making its decision as to the

[3] These techniques, however, require that the object code files be linked with the MSHN Client Library, therefore they require object code files. However, another tool, the Executable Editing Library (EEL) which evolved from the University of Wisconsin's Paradyn project could be used to link an executable with the MSHN Client Library, instead [32].

[4] In modern systems, the choice of computer that is responsible for executing a process often carries with it, implicitly, a choice of file servers and other distributed resources such as networks. Therefore, when we say that MSHN chooses a computer to be responsible for executing a process, the choice of other resources external to that computer may be implicit in that assignment.

assignment of resources, we designed both the MSHN Resource Requirements Database and the MSHN Resource Status Server.

The Resource Status Server is a quickly changing repository that maintains information concerning the current availability of resources. Information is stored in the Resource Status Server as a result of updates from both the MSHN Client Library and the MSHN Scheduling Advisor. The Client Library can update the Resource Status Server as to the currently perceived status of resources, which takes into account resource loads due to processes other than those managed by MSHN. The Scheduling Advisor can provide expected future resource status based upon the resources that it expects will be used by the applications that it assigns. Additionally, the Resource Status Server can statistically process its historic knowledge to make predictions of resource status even further in the future.

As compared to the Resource Status Server, the information maintained by the MSHN Resource Requirements Database changes much more slowly. The Resource Requirements Database is responsible for maintaining information about the resources that are required to execute a particular application. Although the initial MSHN prototype only implements a single source for the information stored in this database (statistically analyzed historical information), we envision that many other on-going research projects will also serve as sources for this information.

MSHN's current source for the information that is maintained by the Resource Requirements Database comes from data collected by the MSHN Client Library when the application was previously executed. Although patterned after SmartNet in this way, and leveraging the concept of compute characteristics that SmartNet pioneered, MSHN does not collect the same information as SmartNet collects. SmartNet's information is coarse-grained; that is, it maintains only the total amount of wall-clock time that is required to execute a program from beginning to end for each particular machine. This measure is sufficient for SmartNet's needs due to the requirements of its intended applications (three phases) and the expected environment (each job has exclusive access to the resources that it is using). However, in MSHN, resources are shared and applications have more phases, so maintaining only this coarse grain information is insufficient. Therefore, the Resource Requirements Database has the ability to maintain very fine grain information collected by the MSHN Client Library. Eventually it is hoped that the Resource Requirements Database can also be populated with information from smart compilers and possibly advice from application writers.

Applications, of course, are needed to test any system. Unfortunately, executables for many different platforms would be needed to test MSHN's ability to manage them in a distributed, heterogeneous environment. Producing such actual applications would require tremendous effort to obtain the source code for numerous applications, some of which may be classified or proprietary, port the source code to the different platforms, and compile and link them. We decided that this effort was better spent on our research system itself, so we looked for another viable solution. One solution that we considered was to use benchmarks, because many of them have already been ported to many different platforms. However, we wanted to make sure that our system could manage a wide variety of applications. We finally settled on writing a general-purpose application emulator whose parameters could be specified to cause it to imitate a wide variety of applications. We discuss the problem of deciding how best to construct such an emulator under the research topics in the next section.

The Client Library, which is linked with each executing MSHN application, informs the Resource Status Server about the current perceived status of the resources that the applications are using. The Scheduling Advisor informs the Resource Status Server only about the load that it expects the processes, which it has scheduled, to place on certain resources. However, neither class of information indicates the condition of resources that no MSHN application is currently using or is planning on using. Therefore, we use a MSHN Application Emulator linked with the Client Library to obtain information about the condition of such resources.

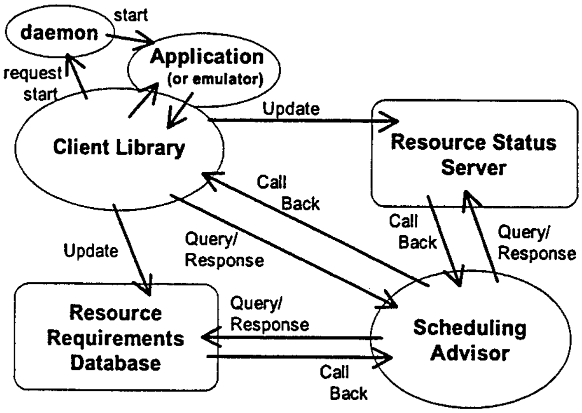


Figure 1 MSHN's conceptual architecture.

MSHN's conceptual architecture is shown in Figure 1. As can be seen in the figure, every application running with MSHN makes use of the MSHN Client Library that intercepts the application's operating system calls. When the Client Library intercepts a request to execute a new application, and that application requires that the MSHN Scheduling Advisor be consulted to determine the resources that the application should use, the Client

Library invokes a scheduling request on the Scheduling Advisor. The Scheduling Advisor queries both the Resource Requirements Database and the Resource Status Server. It uses information that it receives from them, along with an appropriate search heuristic, to determine where the newly requested process should execute. After determining which resources should host the new process, the Scheduling Advisor returns the decision to the Client Library, which, in turn, requests execution of that process through the appropriate MSHN Daemon. The MSHN Daemon invokes the application on its machine. As a process executes, the Client Library updates both the Resource Status Server and the Resource Requirements Database with the current status of the resources and the requirements of the process. Meanwhile, the Scheduling Advisor establishes callbacks with both the Resource Requirements Database and the Resource Status Server. Using callbacks, the Scheduling Advisor is notified in the event that either the status of the resources has significantly changed, or the actual resource requirements are substantially different from what was initially returned from the Resource Requirements Database. In either case, if it no longer appears that the assigned resources can deliver the required QoS, the application must be adapted or terminated. Upon receipt of a callback, the Scheduling Advisor might require that several of the applications adapt so that more of them can receive their requested or desired QoS.

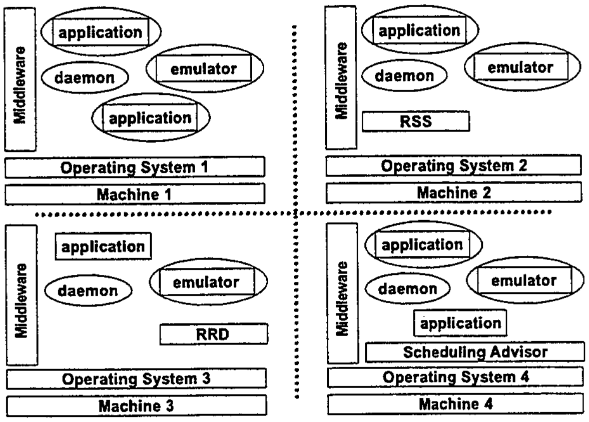


Figure 2 Physical instantiation of the MSHN architecture.

Although all MSHN components could run on the same machine, they can also be distributed and replicated across many different computers using tools such as ISIS, Horus and Ensemble [7][50][49]. Results from control theory will also be useful here in ensuring that the process of replicating and merging components is stable and does not result in oscillation. Additionally, results from control theory must be incorporated into the replicated Scheduling Advisor itself to ensure that modifications requested of adaptive and adaptation-aware applications do not become unstable. MSHN components might even replicate as needed [20][21]. **Figure 2** illustrates a simple instantiation of the MSHN system.

In addition to the components discussed above, we found it convenient to add a MSHN Visualizer that enabled us to examine, for both functional and performance debugging purposes, the current states of the various MSHN components. The MSHN Visualizer captures all significant events within and between the core MSHN components for real-time and post-mortem analysis.

Security within the MSHN architecture has been considered. Policies of interest are:

- Component authentication. This includes authentication of MSHN core components to each other; authentication of resource-based clients to the MSHN core; and authentication of applications to selected MSHN components.
- Hierarchical least privilege. Within the MSHN context, the core components are the most privileged, while user applications are the least privileged.
- Communications integrity and confidentiality. Communications are protected from unauthorized modification and disclosure.
- Access control. Access to MSHN core databases and to job histories may be mediated.

The security architecture creates keyed domains, supporting least privilege, authentication, confidentiality and integrity by using the Common Data Security Architecture facilities for security services and key management[5] [57][58].

2.2.1. The current MSHN architecture. A high level description of the current MSHN architecture is presented. For a more detailed description, we refer the reader to other publications [43]. High-level diagrams are presented for each MSHN component, with arrows indicating the direction of communication or action. In addition to these diagrams, a short description of each component's functions is given. In the description of the MSHN architecture, we represent MSHN components and external components as Unified Modeling Language (UML) actors [8]. The symbols used for this representation are shown in Figure 3. The core MSHN components include the Scheduling Advisor (SA), the Client Library (CL), the Resource Status Server (RSS), the Resource Requirements Database (RRD), the Daemon (D) and the Application Emulator (AE).

[5] As in any RMS, assurance of MSHN's security properties is built on and limited by the effectiveness of the security environment provided by the underlying operating system(s) and hardware base(s).

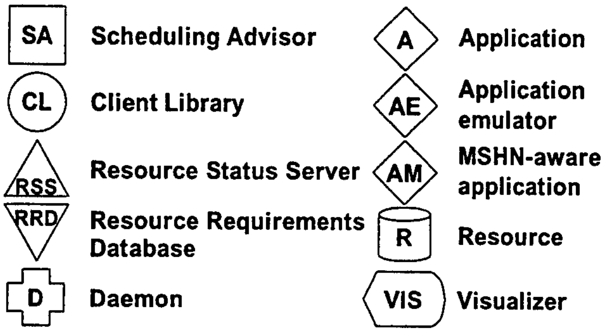
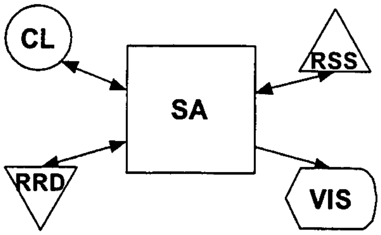
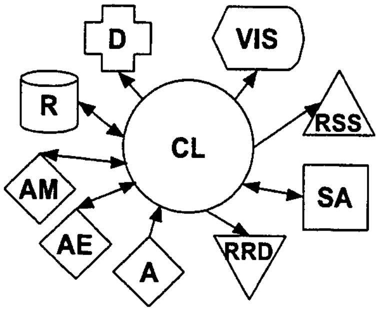


Figure 3 Symbols representing actors in the MSHN architecture.

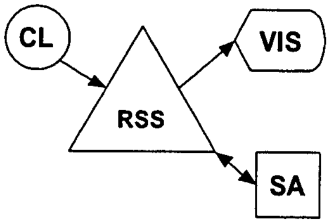
__Scheduling Advisor (SA) functionality__. The primary responsibility of the SA is to determine the best assignment of resources to a set of applications, based on the optimization of a global measure, which we describe in the next section. The SA depends on the RRD and the RSS in order to identify an operating point that optimizes the global measure. It responds to resource assignment requests from the CL. When appropriate, the SA requests application adaptations via the CL. The SA is also responsible for establishing callback criteria (thresholds) with the RSS and RRD. All MSHN components update the MSHN Visualizer with all significant display and post-mortem analysis events.



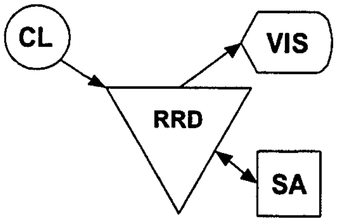
__Client Library (CL) functionality.__ The CL is linked with both adaptive and adaptation-aware applications. It provides a transparent interface to all of the other MSHN components. The CL intercepts system calls to collect resource usage and status information, which is forwarded to the RRD and the RSS. The CL also intercepts calls that initiate new processes (such as `exec()`) and consults the SA for the best place to start that process. It requests (possibly remote) daemons to execute applications based on the SA's advice. The CL invokes adaptation on adaptation-aware applications when notified by the SA via callbacks. One such invocation is the special case of setting emulator parameters.



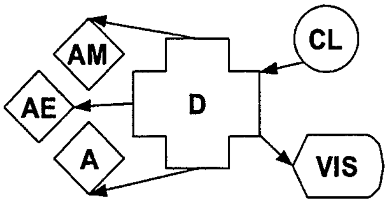
__Resource Status Server (RSS) functionality__. The role of the RSS is to maintain a repository of the three types of information about the resources available to MSHN: relatively static (long-term), moderately dynamic (medium-term), and highly dynamic (long-term) information. The RSS is updated with current data via the CL or through a system administrator. The RSS responds to SA requests with estimates of currently available resources. The SA sets up callbacks with the RSS based on resource availability thresholds and CL update frequency requirements.



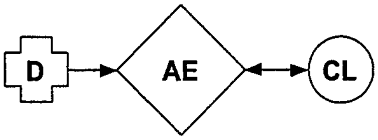
__Resource Requirements Database (RRD) functionality__. The RRD is a repository of information pertaining to the resource usage of applications. The RRD provides this information to the SA. Callbacks to the SA are based on either the occurrence of a threshold violation or update frequency requirements. It is updated by the CL.



__Daemon (D) functionality__. The MSHN Daemon executes on all compute resources available for use by the SA. Its sole purpose is to start applications as requested by the CL. It therefore has the capability and responsibility of initiating the default application emulator at start-up to determine resource status information.



Application Emulator (AE) functionality. The AE emulates a running application by stressing particular resources in the same way as the real application does. The AE serves two purposes: The first is to run simulated applications (that statistically leave the same resource usage footprint of the real applications) without the overhead and uncertainty of actually installing, maintaining, and running that particular application. The second is to be a monitor, in the absence of any other MSHN-scheduled applications. That is, it can determine the status of resources that are not being otherwise used by MSHN-scheduled applications, and therefore not being monitored by an existing CL. The Daemon starts one instance of the AE, by default, at startup. Other instances may be started at any other time through a command interpreter or other application.



3. MSHN Research Issues

In this section of the paper, we describe some of the major issues being investigated by the MSHN team members. We also briefly summarize some of the results to date. Of course, there is not sufficient space to completely describe all of the issues and results in detail, so the reader is also referred to relevant papers on each topic. We have attempted to associate the issues with the component of the MSHN architecture that they most strongly affect. However, certainly many issues that affect the Scheduling Advisor also affect the Resource Status Server and Resource Requirements Database. Additionally, this work is non-orthogonal to research being done by many investigators outside of the MSHN team who are examining such issues as how QoS requirements are derived from smart compilers and how they can be best expressed.

3.1. Scheduling Advisor research issues

In this section we discuss some issues that most strongly affect the Scheduling Advisor. First, we examine how to quantify the needs of all of the processes that require resource allocation by the Scheduling Advisor. Then, we consider the ramifications of not precisely knowing the resource requirements, and consequently, the exact future status of all of the resources. Finally, we discuss the class of heuristics that have thus far been implemented in MSHN and why there is a need for a variety of heuristics.

3.1.1. Optimization criteria. Optimal resource allocation always involves attempting to solve an optimization problem, which is usually NP-complete. SmartNet's primary optimization criterion was to minimize the time at which an application completes, assuming that all of the applications were of a particular form. Later versions of SmartNet also accounted for priorities. MSHN maximizes a weighted sum of values that represents the benefits and costs of delivering the required and desired QoS (including security, priorities, and preferences for versions), within the specified deadlines, if any. We now discuss the effect of each of these attributes on the optimization criteria.

- MSHN's consideration of security as an optimization criterion allows the trade-off of security with other QoS constraints when there are insufficient resources to complete all requests. This is done in a fashion similar to other recent projects [45]. MSHN associates a cost to security levels that varies, depending upon which resources are being used to obtain a given level of security (for more details on security viewed as a QoS parameter, see section 3.2).
- MSHN attempts to account for both preferences for various versions and priorities. That is, when it is impossible to deliver all of the most preferred information within the specified deadlines due to insufficient resources, MSHN's optimization criteria are designed to favor delivering the most preferred version to the highest priority applications.
- In MSHN's optimization criteria, deadlines can be simple or complex. That is, sometimes a user will be satisfied if a result is received before a specific time. Other times, a user would like to associate a more general benefit function, which would indicate that information might have different values based upon when it is received.

Further information about MSHN's optimization criteria can be found elsewhere [22][30].

In addition to a cost function that is optimized, optimization problems usually have a set of constraints that must be met in order for a solution to be viable. The constraints of a resource allocation optimization problem are that the resources allocated to meet the needs of the processes must be less than or equal to the available resources at any point in time. The actual inequalities required not only depend upon the QoS constraints, but also upon the sharing policies used by the local operating systems and network protocols, and upon the granularity with which both those policies and resource usage should be known (see Granularity Issues in Section 3.2).

3.1.2. Inexact knowledge of job resource usage. Even if it is possible to find a perfect solution to the optimization problem that is posed by instantiating the constraints and optimization criteria to the current situation, the expected resource usage of any given application is often only an estimate. In real-time systems, the worst case estimate is often used to assign resources to processes; however, many other systems use the mean expected resource usage. Our recent analysis has revealed that using the mean will cause the actual run-time to be generally underestimated and that a better assignment can be made if both the mean and distribution of the expected resource usage is accounted for, when appropriate [28].

This leads to another question concerning whether the extra complexity involved in using a sophisticated heuristic will yield a better schedule than using a simple heuristic if the actual variance of run-times is large, and scheduling is done using the mean, or both the mean and the distribution. Our recent results in this area have shown that, in many cases, complex heuristics can determine schedules that, when executed, sometimes perform much better than the schedules derived from very simple heuristics, even when the variance is large. However, sometimes very simple heuristics perform just as well as the more complex ones. The difference in quality of the schedules produced by the various heuristics was found to be closely correlated with the type of heterogeneity in a system. For example, when both the machine and application heterogeneity is very low, a simple heuristic performs just as well as more complex ones. Several papers have described our results concerning this research [2][3][10][40].

3.1.3. Performance of search algorithms. SmartNet's organization leveraged the idea of independence of search algorithms and optimization criteria. That is, most heuristics for searching the space of mappings can be modified to search for solutions to different optimizations within the same space. For example, Dantzig's Simplex Method is useful with all problems whose optimization criteria and constraint inequalities can be stated using only linear combinations of the variables. Sometimes, many different heuristics will work, but, depending upon the characteristics of a given problem, certain heuristics may be preferable to others. For example, the MSHN team has obtained extensive results identifying the regions of heterogeneity where certain heuristics perform better than others for maximizing throughput by minimizing the time at which the last application, of a set of applications, should complete [2][3][10][40]. Re-targeting of these heuristics to other optimization criteria is currently underway.

Additionally, MSHN team members have performed extensive research into accounting for dependencies between applications or processes that make up a single application [40][47][48][54]. This includes promising results from investigating data dependencies and mapping of iterative applications [1][4][5][6][11].

3.2. Resource Status Server and Resource Requirements Database research issues

Part of the MSHN team's investigation has been aimed at determining what information should be stored in the Resource Requirements Database and maintained by the Resource Status Server. First, a taxonomy for the types of information that could be stored there was required. We discuss this taxonomy below. We also discuss the impact that viewing security as a QoS has on these two MSHN components. Finally, one of the most important issues in designing effective RMSs is determining the level of granularity of information that must be maintained concerning the status of resources and the requirements of applications. We now discuss each of these issues in somewhat more detail and refer the interested reader to relevant publications.

3.2.1. A taxonomy. The MSHN team has formulated a three-part taxonomy for classifying systems. The three different components include methods for describing the applications, the computing environment, and the mapping strategy that is used. Some of the relevant characteristics that need to be instantiated concerning each application include

(i) Its size, that is the number of tasks or sub-tasks associated with it.

(ii) Whether the sub-tasks are independent of one another or, if they are dependent, the types of dependencies.

(iii) The I/O distributions of the application and the sources of the I/O, i.e., whether it performs all input in the beginning and all output at the end or whether one or the other is performed continually throughout the lifetime of the processes and whether the input data is obtained through interacting with a person or some other source that has highly variable response times.

(iv) The deadlines and other QoS requirements, including security, if any, associated with the applications and/or the subtasks that comprise the application.

Similarly, the computing environments and mapping strategies have numerous, hierarchically characterizable, attributes that are more fully documented in other publications [9].

3.2.2. Security as a quality of service. Security in the context of QoS is a current research area [34][45]. The security capabilities of resources and security requirements of applications must influence the assignment of applications to resources. We can obtain information concerning the user security requirements from the Resource Requirements Database and information concerning the security capabilities of the resource from the Resource Status Server. For example, if the output of an application must be encrypted using a particular algorithm, with a key size chosen within a particular range, then that requirement must be stored in the Resource Requirements Database along with the amount of data that must be encrypted. Also, the Resource Status Server must know whether each particular computing resource is capable of performing the required cryptographic algorithm and the cost, in terms of run-time per byte, for example, of encrypting the data. Members of the MSHN team have developed an initial framework, which they are currently refining, for characterizing the overall security attributes of a network and for determining a cost and benefit value for providing required and preferred security to an application [26][27][33][34].

3.2.3. Granularity issues. Another very important question that concerns both the Resource Requirements Database and the Resource Status Server has to do with how much detail should be maintained concerning the status of resources and the requirements of applications. Obviously, while a very accurate, detailed set of information might prove quite useful to the scheduling algorithms, it would be at the least very expensive and difficult to collect if not expensive to process within the algorithm itself.

The MSHN team has obtained initial estimates for the overhead of capturing system calls to determine the cost of collecting various granularities of such information [44]. Members of the team are currently using this technique to record fine-grained information for a program that analyzes air tasking orders and will report both the information concerning the resources that were used, as well as the overhead involved in collecting the resource usage information [41].

In addition to the cost associated with collecting fine-grained information concerning applications' use of resources, there is the question of how much information is sufficient. Current experiments of the MSHN team focus on determining whether fairly simple models can be used to predict the relative performance of application/resource assignments. To perform realistic experiments, the team has built an initial application emulator (see below) and is actually executing it with different parameters on different systems, using all possible configurations to compare the actual received QoS to the predicted QoS. Thus far we have determined that the Resource Status Server must, directly or indirectly, contain information concerning whether native threads are supported by the operating system. If this information is not maintained, the scheduling algorithm, which must choose between two platforms that are identical except for the operating system version that they execute, may assign a process which could be handled better by one platform to the other. Similarly, the Resource Requirements Database must indicate whether or not the application is multi-threaded and the number and nature of threads that it uses. Information concerning these results can be found in other publications [12].

3.3. Application Emulator research issues

The MSHN team is designing and implementing an application emulator for two different reasons. One reason is that it is needed within the MSHN architecture to monitor the end-to-end status of the resources. The other reason is to be able to easily construct a very large suite of application emulators that place loads on resources in the same way that the actual applications would. When used in conjunction with resource usage measurements from linking actual applications to MSHN's Client Library, the MSHN Application Emulator can be used to emulate the execution of the actual applications without requiring the applications to actually be ported to many different platforms. The obvious advantage of using such an application emulator, rather than porting the applications themselves, is to enable the MSHN researchers to test their architecture more quickly under many different situations.

To meet the first purpose of the MSHN Application Emulator, we first had to define the meaning of loading resources for various resources. Percentages cannot be used, as they are not transferable between either computing platforms or network media. Rather, each category of resource was identified and units that can be most easily translated between different platforms, such as FLOPS and bytes/sec, were chosen to quantify resource use. Also recognized at this stage was the need to have both multi-threaded and non-multi-threaded application emulator capability. Finally, not only can a single application be comprised of multiple threads, but it can also be comprised of multiple heavy-weight processes.

When designing the Application Emulator to meet both of its requirements, we recognized that distributions reflecting communication and computation alone were insufficient; conditional probabilities were required. That is, many times the purpose of one process sending a message to another process is so that the receiving process will perform work on behalf of the sending process. Therefore, we designed our most general emulator to also have the capability of sending work-bearing messages.

To this end, we have completed an initial implementation of an application emulator that we have used for our granularity research and are testing the more general application emulator. Documentation concerning both of these application emulators can be found elsewhere [12] [15].

3.4. Client Library research issues

The research issues having to do with the Client Library component involve both mechanism and policy. The mechanism issues have to do with how to transparently link the Client Library with applications. Previous research in the areas of process migration and tools for debugging parallel and distributed programs provide us with easy solutions, as mentioned earlier. Therefore, the only issue that remains is how best to transparently determine the end-to-end availability of resources. First, simply determining that the Client Library could perform this functionality better than providing the functionality external to the applications themselves is an important contribution. However, determining the average end-to-end availability of a network resource is not a trivial problem. The MSHN team's initial progress in this area has already been detailed elsewhere [30][31][44].

4. Summary and future work

In this paper we summarized the purpose of a resource management system (RMS) in general and the research goals of one particular experimental RMS, the Management System for Heterogeneous Networks (MSHN). Motivation was provided for all of the major components of MSHN, and the architecture that contains those components was explained. Some of the research questions that the MSHN researchers are seeking answers to were described. References were provided that enable the reader to better understand MSHN, and to learn more about the MSHN experiments. There are many other interesting RMS research projects in progress today, but space permitted us to survey only a few of them. In addition to continuing the on-going experiments described in the paper, future MSHN investigation will focus on (i) reaching a better understanding of the level of granularity obtainable from applications and the level required to perform sufficiently good resource assignment; (ii) more detailed characterization of security costing and metrics; and (iii) determining the best search algorithms to use for the MSHN optimization criteria under various conditions.

Acknowledgments – The authors thank Shushanna St. John and LCDR Wayne Porter for their comments.

References

[1] A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra, "A unified resource scheduling framework for heterogeneous computing environments," *Proc. 8th IEEE Heterogeneous Computing Workshop*, April 1999.

[2] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Sept. 1997.

[3] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *Proc. 7th IEEE Heterogeneous Computing Workshop*, March 1998, pp. 79-87.

[4] P. B. Bhat, V. K. Prasanna, and C.S. Raghavendra, "Adaptive communication algorithms for distributed heterogeneous systems," *Proc. IEEE Intl. Symp. High Performance Distributed Computing,* July 1998, pp. 310-321.

[5] P. B. Bhat, V. K. Prasanna, and C.S. Raghavendra, "Block-cyclic redistribution over heterogeneous networks," *Proc. ISCA Intl. Conf. Parallel and Distributed Computing Systems*, Sept. 1998, pp. 242-249.

[6] P. B. Bhat, V. K. Prasanna, and C.S. Raghavendra, "Efficient collective communication in distributed heterogeneous systems," *Proc. IEEE Intl. Conf. Distributed Computing Systems*, 1999, to appear.

[7] K. Birman, "Replication and fault-tolerance in the ISIS system," *10th ACM Symposium on Operating Systems Principles*, Dec. 1985, pp. 79-86.

[8] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, Reading, MA, 1999.

[9] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *Proc. IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1998, pp. 330-335 (included in the proceedings of the *7th IEEE Symposium on Reliable Distributed Systems*, 1998).

[10] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," *Proc. 8th IEEE Heterogeneous Computing Workshop*, April 1999, to appear.

[11] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms," *The Journal of Supercomputing*, Vol. 12, No. 4, Oct. 1998, pp. 387-406.

[12] P. Carff, *Granularity*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, March 1999.

[13] P. Chandra, A. Fisher, C. Kosak, T. S. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang, "Darwin: Resource Management for Value-Added Customizable Network

Service," *Proc. 6th IEEE International Conference on Network Protocols*, October 1998, pp. 177-188.

[14] K. Czajkowski, I. Foster, C. Kesselman, N. Karonis, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," *Proc. Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62-82.

[15] T. Drake, *A Load Emulator Toolkit and Analysis of HiPer-D Resource Requirements*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, June 1999.

[16] S. Fickas, and M. S. Feather, "Requirements Monitoring in Dynamic Environments," *Proc. 2nd IEEE Intl. Symposium on Requirements Engineerings*, March 1995, pp. 140-147.

[17] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," *Proc. 6th IEEE Symp. on High-Performance Distributed Computing*, 1997, pp. 365-375.

[18] I. Foster, and C. Kesselman, "The Globus project: a status report, " *Proc. 7th IEEE Heterogreneous Computing Workshop*, 1998, pp. 4-18.

[19] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Kieth, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *Proc. 7th IEEE Heterogeneous Computing Workshop*, March 1998, pp. 184-199.

[20] D. Hensgen, *Squads: Server Groups that Dynamically Adapt to Improve Performance*, Ph. D. Dissertation, Department of Computer Science, University of Kentucky, 1989.

[21] D. Hensgen, and R. Finkel, "Dynamic server squads in Yackos," *Proc. Workshop on Experiences with Building Distributed and Multiprocessor Systems*, Oct. 1989, pp. 73-90.

[22] D. Hensgen, T. Kidd, H. J. Siegel, J. K. Kim, D. St. John, C. Irvine, T. Levin, V. Prasanna, and R. Freund, *A performance measure for distributed heterogeneous networks based on priorities, deadlines, versions, and security*, Technical Report, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, Feb. 1999.

[23] D. Hensgen, L. Moore, T. Kidd, R. F. Freund, E. Keith, M. Kussow, J. Lima, and M. Campbell, "Adding rescheduling to and integrating Condor with SmartNet," *Proc. 4th IEEE Heterogeneous Computing Workshop*, April 1995, pp. 4-11.

[24] J. Huang, R. Jha, W. Heimerdinger, M. Muhammad, S. Lauzac, B. Kannikeswaran, K. Schwan, W. Zhao and R. Bettati, "RT-ARM: a real-time adaptive resource management system for distributed mission-critical applications", *Workshop on Middleware for Distributed Real-Time Systems*, 1997, pp. 179-186.

[25] O. Ibarra and Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, Vol. 24, No. 2, 1977, pp. 280-289.

[26] C. Irvine and T. Levin, *A note on mapping user-oriented security policies to complex mechanisms and services*, Technical Report, Department of Computer Science, Naval Postgraduate School, Monterey, CA, in progress.

[27] C. Irvine and T. Levin, *Toward a taxonomy and costing method for security metrics*, Technical Report, Department of Computer Science, Naval Postgraduate School, Monterey, CA, in progress.

[28] T. Kidd and D. Hensgen, *Why the Mean is Inadequate for Making Scheduling Decisions*, Technical Report, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Jan. 1999.

[29] T. Kidd, D. Hensgen, R. Freund, and L. Moore, "SmartNet: a scheduling framework for heterogeneous computing," *Proc. 2nd Intl. Symposium on Parallel Architectures, Algorithms, and Networks*, June 1996, pp. 514-521.

[30] J. P. Kresho, *Quality Network Load Information Improves Performance of Adaptive Applications*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Sept. 1997.

[31] J. P. Kresho, D. Hensgen, T. Kidd, and G. Xie, "Determining the accuracy required in resource load prediction to successfully support application agility," *Proc. 2nd IASTED Intl. Conf. European Parallel and Distributed Systems*, July 1998, pp. 244-254.

[32] J. Larus and E. Schnarr, "EEL: machine-independent executable editing," *SIGPLAN PLDI 95*, 1995, pp. 291-300.

[33] T. Levin and C. Irvine, *An approach to characterizing resource usage and user preferences in benefit functions*, Technical Report, Department of Computer Science, Naval Postgraduate School, Monterey, CA, in progress.

[34] T. Levin and C. Irvine, *Quality of security service in a resource management system benefit function*, Technical Report, Department of Computer Science, Naval Postgraduate School, Monterey, CA, in progress.

[35] J. W. S. Liu, K. Nahrstedt, D. Hull, S. Chen, and B. Li, *EPIQ QoS Characterization, Draft Version*, July 1997, http://epiq.cs.uiuc.edu/files/qos-970722.pdf.

[36] M. Livny, M. Litzkow, T. Tannenbaum, and J. Basney, "Checkpoint and migration of UNIX processes in the Condor distributed processing system," *Dr Dobbs Journal*, Feb. 1995, pp. 40-51.

[37] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken, "Specifying and measuring quality of service in distributed object systems," *Proc. 1st Intl. Symposium on Object-Oriented Real-Time Distributed Computing*, April 1998, pp. 20-22.

[38] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," *Proc. 8th IEEE Heterogeneous Computing Workshop*, April 1999, to appear.

[39] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. Webster, ed., John Wiley & Sons, New York, NY, to appear 1999.

[40] M. Maheswaran and H. J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," *Proc. 7th IEEE Heterogeneous Computing Workshop*, Mar. 1998, pp. 57-69.

[41] N. W. Porter, *Resource Requirement Analysis of GCCS Modules and EADSim and Determination of Future Adaptivity Requirements*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, June 1999.

[42] A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza, J. Simon, T. Römke, and F. Ramme, "The MOL project: an open extensible metacomputer," *Proc. 6th IEEE Heterogenous Computing Workshop*, April 1997, pp. 17-31.

[43] D. St. John, S. Kidd, D. Hensgen, T. Kidd, and M. Shing, *Experiences using semi-formal methods in MSHN*, Technical Report, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Feb. 1999.

[44] M. C. L. Schnaidt, *Design, Implementation, and Testing of MSHN's Application resource Monitoring Library*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Dec.1998.

[45] P. Schneck and K. Schwan. "Dynamic authentication for high-performance networked applications," *Proc. 6th IEEE/IFIP Intl. Workshop on Quality of Service*, May 1998, pp. 127-136.

[46] J. Sydir, B. Sabata, and S. Chatterjee, " QoS middleware for the next-generation Internet," position paper, *Proc. NASA/NREN Quality of Service Workshop*, Aug. 1998, pp. 25-27.

[47] M. Tan and H. J. Siegel, "A stochastic model for heterogeneous computing and its application in data relocation scheme development," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 11, Nov. 1998, pp. 1088-1101.

[48] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, Aug. 1999, pp. 857-871.

[49] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr, *Building adaptive systems using Ensemble*, Cornell University Technical Report, TR97-1638, July 1997.

[50] R. van Renesse, K. Birman, and S. Maffeis, "Horus, a flexible group communication system," *Communications of the ACM*, April 1996, pp. 76-83.

[51] R. van Renesse and A. S. Tanenbaum, "Distributed operating systems," *ACM Computing Surveys*, Vol. 17, No. 4, Dec. 1985, pp. 419-470.

[52] R. Vanegas, J. A. Zinky, J. P. Loyall, D. A. Karr, R. E. Schantz, and D. E. Bakken. "QuO's runtime support for quality of service in distributed objects," *Proc. IFIP Intl. Conf. on Distributed Systems Platforms and Open Distributed Processing* , Sept. 1998, pp. 207-222.

[53] J. Walpole, C. Krasic, L. Liu, D. Maier, C. Pu, D. McNamee, and D. Steere, "Quality of service semantics for multimedia database systems," *Proc. Data Semantics 8: Semantic Issues in Multimedia Systems IFIP TC-2 Working Conference*, Jan. 1999, pp. 393-412.

[54] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 199, pp. 8-22.

[55] L. R. Welch, B. Ravindran, B. A. Shirazi, and C. Bruggeman, *Specification and modeling of dynamic, distributed real-time systems*, Technical Report Number TR-CSE-98-003, Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX, Sept. 1998.

[56] L. R. Welch, B. Ravindran, B. A. Shirazi, and C. Bruggeman, "DeSiDeRaTa: QoS Management Technology for Dynamic, Scalable, Dependable, Real-time Systems," *Proc. 15TH IFAC Workshop on Distribute Computer Control Systems*, Sept. 1998, pp. 7-12.

[57] R. E. Wright, *Management System for Heterogeneous Networks Security Services*, Thesis, C4I Academic Group, Naval Postgraduate School, Monterey, CA, June 1998.

[58] R. E. Wright, D. J. Shifflett, and C. E. Irvine, "Security for a virtual heterogeneous machine," *Proc. 14th Computer Security Applications Conference*, Dec. 1998, pp. 167-177.

Biographies

Debra Hensgen received her PhD in the area of Distributed Operating Systems from the University of Kentucky. She is an Associate Professor in the CS Department at The Naval Postgraduate School. She has co-authored numerous papers about the Concurra toolkit for automatically generating safe, efficient concurrent code, the Graze parallel processing performance debugger, the SAAM path information base, and the SmartNet and MSHN Resource Management Systems.

Taylor Kidd obtained his PhD from the University of California at San Diego in 1991. His interests include both theoretical and applied distributed computing. He led the research component of the SmartNet team at NRaD and instigated a number of important advances to the SmartNet Scheduling Framework. He is a co-PI for the DARPA-sponsored MSHN project and a co-investigator on the DARPA-sponsored SAAM Project.

David St. John is the head of staff at the Heterogeneous Network & Computing Laboratory, Naval Postgraduate School. He has over six years experience in object-oriented software development in the areas of distributed computing, process control, sensor collection, and Internet transaction processing systems. He is a member of IEEE and IEEE Computer Society. He received a BSME with High Honors from the University of Florida and an MSE from the University of California, Irvine.

Matt Schnaidt earned his professional engineering license while serving as the Battalion Adjutant. He received an MS from the Computer Science Department of the Naval Postgraduate School in 1998. Currently, Major Schnaidt is working on the Battle Management Command, Control and Communication component of the National Missile Defense Program.

H. J. Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is an IEEE Fellow and an ACM Fellow. He received two BS degrees from MIT, and the MA, MSE, and PhD degrees from Princeton University. He has coauthored over 250 technical papers, was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was an editor of the IEEE Transactions on Parallel and Distributed Systems.

Tracy D. Braun is a PhD student and Research Assistant at Purdue University. He received his BSEE with Honors and High Distinction from the University of Iowa in 1995. In 1997, he received his MSEE from the School of Electrical and Computer Engineering at Purdue. He is a member of IEEE, IEEE Computer Society, and Eta Kappa Nu honorary society. His research interests include parallel algorithms, heterogeneous computing, computer security, and software design.

Muthucumaru Maheswaran is an Assistant Professor in the Department of Computer Science at the University of Manitoba, Canada. He received a BSc degree from the University of Peradeniya, Sri Lanka and the MSEE and PhD degrees from Purdue University. He received a Fulbright scholarship to pursue his MSEE degree at Purdue University. His research interests include computer architecture, distributed computing, heterogeneous computing, and resource management systems for metacomputing.

Shoukat Ali is an MSEE student at the School of Electrical and Computer Engineering at Purdue University. His main research topic is dynamic mapping of meta-tasks in heterogeneous computing systems. He has held teaching positions at Aitchison College and Keynesian Institute of Management and Sciences, both in Lahore, Pakistan. Shoukat received his BS degree from the University of Engineering and Technology, Lahore, Pakistan in 1996. His research interests include computer architecture, parallel computing, and heterogeneous computing.

Jong-Kook Kim is an MSEE student and Research Assistant in the school of Electrical and Computer Engineering at Purdue University, currently working on the DARPA/ISO sponsored BADD program. He received his BSEE from Korea University, Korea. He served in the ROK Army working with the US Army on the Theater Automated Command and Control Information Management System and received the US Army Commendation Medal. His research interests include heterogeneous computing and performance measures for distributed systems.

Cynthia E. Irvine is Director, Naval Postgraduate School Center for INFOSEC Studies and Research and an Assistant Professor of Computer Science at the Naval Postgraduate School. Dr. Irvine holds a PhD from Case Western Reserve University. She has over twelve years experience in computer security research and development. Her current research centers on architectural issues associated with applications for high assurance trusted systems, security architectures combining popular commercial and specialized multilevel components, and the design of multilevel secure operating systems.

Timothy Levin is currently doing research at the Naval Postgraduate School. He received a BS in Computer and Information Science from the University of California at Santa Cruz, 1981. His secure system work includes design of security features, and formal verification and formal covert channel analysis of an A1 operating system, and enterprise security features for a commercial relational database system. He has been certified by the NSA as a Vendor Security Analyst, for participation in their Trusted Product Evaluation Program.

Richard Freund is a founder and CEO of NOEMIX, a San Diego based startup to commercialize distributed computing technology. Freund is also one of the early pioneers in the field of distributed computing, in which he has written or co-authored a number of papers. In addition he is a founder of the Heterogeneous Computing Workshop, held each year in conjunction with IPPS/SPDP. Freund won a Meritorious Civilian Service Award during his career as a government scientist.

Matt Kussow, B.S.C.S., has 12 years of experience in software development, research, design, process analysis, and project management. Currently he holds the position of Vice President of Product Development at Noemix. He has extensive experience in designing and developing software for high performance computing, parallel algorithms, network computing, and database systems.

Michael Godfrey, B.S. C.I.S, UCSD Java Certified, has over 6 years of software research, design, and development experience with high performance computing, secure world wide web, health care, and data base systems for Science Applications International Corporation (SAIC) and Noemix, Inc. He has developed systems level applications on a variety of UNIX and Win32 platforms.

Alpay Duman is a LTJG in the Turkish Navy. He graduated from the Turkish Naval Academy with a BS in Operations Research with honors. He received his MSCS degree in the area of Systems Design and Architecture from the Naval Postgraduate School. He is currently a systems engineer at Turkish Navy Software Development Center working on a CORBA based communication infrastructure for Command Control Systems.

Paul F. Carff, LT US Navy, is a Masters student in the Computer Science Department at the Naval Postgraduate School, working on the Management System for Heterogenous Systems (MSHN). He received a BS in Engineering Physics from Santa Clara University in 1991. Prior to coming to Naval Postgraduate School, LT Carff served 3 years as a Nuclear Power Officer aboard the USS Salt Lake City (SSN 716).

Shirley Kidd is one of the supporting staff at the Heterogeneous Network & Computing Laboratory. She has 4 years of experience in the aerospace industry and another 6 years at a commercial marketing company during which she worked in both industries as a programmer analyst. She has a BS in Applied Mathematics from the University of California, San Diego.

Viktor K. Prasanna (V.K. Prasanna Kumar) is a Professor in the Department of Electrical Engineering Systems, University of Southern California, Los Angeles. He obtained his PhD in Computer Science from Pennsylvania State University in 1983. He has published and consulted for industries in parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision. He serves on the editorial boards of the Journal of Parallel and Distributed Computing and IEEE Transactions on Computers. He is the founding Chair of the IEEE Computer Society Technical Committee on Parallel Processing, and a Fellow of the IEEE.

Prashanth B. Bhat is a PhD candidate in Computer Engineering at the University of Southern California, Los Angeles. He received his B.Tech. degree in Computer Engineering from the Karnataka Regional Engineering College, India, in 1992. He received his ME degree in Computer Science and Engineering from the Indian Institute of Science, Bangalore, in 1994. His research interests include scheduling techniques for parallel and distributed systems, High Performance Computing and parallel computer architecture. During the summer of 1998, he was a research intern at Hewlett-Packard laboratories, Palo Alto.

Ammar Alhusaini is a PhD student in the Electrical Engineering Department at the University of Southern California. His main research interest is task scheduling in heterogeneous environments. He received an MS degree in computer engineering from the University of Southern California in 1996. He is a member of IEEE, IEEE Computer Society, and ACM.

A Note on Mapping User-Oriented Security Policies to Complex Mechanisms and Services[1]

Cynthia Irvine
Naval Postgraduate School
Monterey, CA

Tim Levin
Anteon Corporation
Monterey, CA

Abstract. *The quality of service framework in a heterogeneous computer network environment may provide users and applications with a wide range of security mechanisms and services. We propose a simplified user security interface and a method for mapping this interface to complex underlying security mechanisms and services. Additionally, we illustrate a mechanism for mapping multiple security policies to the same user security interface.*

1 Introduction

In a heterogeneous computer network[2], the user can be presented with a wide range of security services and enforcement mechanisms [5] instituting security policies from various security domains. The security domains can be geographically diverse (e.g., subnets traversed to a remote internet destination) and layered (e.g., application security policies versus network security policies). The problem of mapping security mechanisms between different network layers is identified in the literature (e.g., see [3] [10]), as is the composition of policies and policy domains (e.g., see [2] [6] [11]). However, the problem remains as to how users and administrators can understand and easily interact with a wide range of security services and mechanisms. This note address the translation of a simplified user abstraction of security to detailed underlying mechanisms, such that users can be presented with a coherent user-level view of available security options.

2 User Security Interface

In the network computing context, users may request the execution of "tasks," which are scheduled by an underlying control program (e.g., a Resource Management System, "RMS") to execute on local or remote computing resources. The execution of a task may access a variety of network resources, such as: local I/O device bandwidth, internetwork bandwidth; local and remote CPU time; local, intermediate (e.g., routing buffers) and remote storage. Each resource may have its own security constraints. One cannot expect users or even application developers to understand the implications of the detailed interfaces of all of these mechanisms. Therefore a simplified, generalizable user-interface is called for.

We present a framework for mapping a simple user interface to an arbitrarily complex set of detailed security mechanisms. We will use the following simple user interface for illustration,

[1] Funded through MSHN, a DARPA/QUORUM project.

[2] A network comprising a variety of software and hardware implementations for processing, networking and storage.

however other simple taxonomies might suffice[1]. We envision a QoS-like interface in which the user may specify the degree or "level" of security service, in general, that is to be applied to the processing of the network task. Such a level might be as simple as:

user_security_level ::= [high | medium | low]

Thus, a user QoS request might appear like this:

QoS Request ::= task_specifier, user_security_level, performance_vector, other_factors

2.1 Application and System Security

Various quality of service approaches are including security as one of the vectors of service provided to the user [4] [9] [12] [14] . It is apparent that, if a QoS system is going to provide choices to the user with respect to security, the underlying mechanisms need to provide variable security, and that the network security policy(s) need to allow security to vary.

However, computer security has been envisioned traditionally at the *system* level [1] [2] . Users and applications were constrained by underlying security mechanisms to behave in ways that conformed to the *system* security policy, and system security policies did not allow the security requirements to vary. For present-day network security, considering the network and the OS(s) to be the "system," there has been some shift of emphasis from *system* security to *application* security. That is, each application (e.g., an email program) may present a security environment to its users, and is responsible for protecting the user's rights and data in that environment and in the network. We believe that the needs for application-level security must be accommodated; however, network system security policies cannot be ignored in the process, rather, different levels of policy must work in harmony. Thus, given that the over-arching network *system* security policy demands some minimum degree of system security policy enforcement, application-level selections for quality of security service may be provided to users to any degree of security over and above those system minimums. That is to say, an application can always provide more security, than the minimum required by the base system security policy, while still complying with that policy. Similarly, application enforcement of user security *maximums* might be possible, e.g., to limit processing expenditures, if those maximums are within the bounds of the underlying security policy(s).

We refer to services and mechanisms that allow a range of security behaviors as "variant." Variant security mechanisms may be used within a resource management context, for example, to effect adaption to varying network conditions. Also, if the policy mechanism is variant, the control program may offer quality of security service choices to the users and their network tasks.

2.2 Security Terminology

Before discussing the mapping of a simplified user security interface to complex underlying mechanisms, some security mechanism terminology is presented (see [5] for further explication).

Users and applications on the network are presented with various security *services* (e.g., data-flow confidentiality, non-repudiation). A security service may be used to implement one or more secu-

[1] TCSEC evaluation classes or Common Criteria profiles could be used

rity *policies* (organizational or automated [15]), which are in turn implemented by one or more security *mechanisms*. As described above, some mechanisms provide fixed services, and some are *variant* [1]. Additionally, the RMS may make choices for the user regarding variant security mechanisms, as part of its schedule formulation or adaptive re-scheduling.

Each security mechanism is associated with a *service area*, which indicates the general topographical component of the network in which the security or protection is effective. We identify three service areas: end system (e.g., a client or server system), intermediate node (e.g., routers, switches), and network connection (i.e., the "wire" connecting various systems and nodes). Security mechanisms associated with end systems and intermediate nodes protect resources (e.g., data and programs) that are associated with a node or system; for network connections, we are concerned with mechanisms for protecting information that is physically in transit.

2.3 Mapping User Security Interface

The elements of the simple user interface are mapped to detailed mechanism invocations via a *translation matrix.* Table 1 shows a mapping of our example user security scale (viz, low,

Table 1: Example User Security Translation Matrix

		User Security Scale		
Security Service	Service Area	Low	Medium	High
Data Confidentiality	ES	none	OS access controls	B3-level DAC
Data Integrity	Wire	none	DES 56-bit key	DES 128-bit key
Login Authenticity	ES	OS I & A	B1-level I & A	Public key certificates
Message Nonrepudiation	ES	none	OS auditing	Digital Notary Service

[1] Variant mechanisms offer the user various "degrees," or strengths, of security (viz., over and above some minimum requirement).

Table 2: Security Translation Matrix with Network Modes

			User Security Scale		
Security Service	Service Area	Network Mode	Low	Medium	High
Data Confidentiality	ES	normal	none	OS access controls	B3-level DAC
		impacted	none	OS access controls	OS access controls
		under attack	OS access controls	B3-level DAC	B3-level DAC
Data Integrity	Wire	normal	none	DES 56-bit key	DES 128-bit key
		impacted	none	none	DES 56-bit key
		under attack	DES 56-bit key	DES 56-bit key	DES 128-bit key
Login Authenticity	ES	normal	none	B1-level I & A	Public key certificates
		impacted	none	B1-level I & A	OS-level I & A
		attack	OS I & A	B1-level I & A	Public key certificates
Message Nonrepudiation	ES	normal	none	OS auditing	Digital Notary Service
		impacted	none	none	OS-level auditing
		under attack	OS auditing	Digital Notary Service	Digital Notary Service

medium, high) to a heterogeneous network which has several *variant* security services. Each level in the user security scale is characterized by one or more mechanisms for each security service. Also, a particular security mechanism may be mapped to more than one user security level, e.g., in Table 2 , 56-bit keyed DES is the mechanism to satisfy data integrity services in the network attack mode for both low and medium user security levels.

In this example, the network has end systems with both simple OS discretionary access control (DAC) and with class B3 evaluated [1] DAC, indicating that the system policy allows OS-level DAC to be enforced with a range of mechanisms. There are also a variety of integrity, authenticity and nonrepudiation services available. With this mapping, the user is not offered all combinations of variant services; instead, the security administrator or system security engineer has pre-selected various specific mechanisms and settings that map to the three choices offered to the user. The idea here is to provide the user/task with a virtual network in which all elements posses consistent assurance qualities, e.g., effectiveness and/or worthiness. Thus, the network security architecture is coherent with respect to each of the service requirements and has no weak link. These example mappings illustrate mechanisms to govern users at the *system* level; mapping pre-selections could be made also at the *application* level, but that is not illustrated here.

This type of translation matrix can be used to both: (1) translate abstract levels or scales of security services to specific settings in the underlying mechanisms (as is illustrated above), and (2) given a set of security mechanisms (e.g., from a distributed system), derive the abstract level of service that is available (e.g., the greatest lower bound).

Thus, users can indicate the desired security *degree or "level"* of their connection, perhaps as part of a QoS request (see Section 2 on page 1). The underlying RMS or control program would be responsible for assigning security services and resources to the user that would meet the security profile indicated by the translation matrix. If corresponding services or resources could not be found to meet the user request, then the RMS would need to negotiate different degrees of service with the user, or perhaps use a default translation.

2.4 System Architecture

The translation matrix can be implemented in a variety of ways. For example, a globally-accessible directory could be managed by a security policy server, and be accessed by the RMS as needed to translate user requests. Alternatively, the matrix could be implemented as an RMS internal table, and managed by an RMS administrative tool.

2.5 Alternative Frameworks

As an alternative to the highly abstract user interface described here, detailed numeric measurements can be applied to each mechanism. Novell defines a security taxonomy within its crypto environment [7], with numeric security-strength indicators for the various components. Wang and Wulf [16] have organized a security taxonomy in a hierarchical fashion to provide a detailed metric for security services. Such a system could present users with a numbering system with which to indicate the desired strength of each security mechanism, and present summary numbers to indicate the overall strength of certain subsystems or sub-networks. However, we feel that much work needs to be done to standardize such metrics, and to educate users as to their meaning.

3 Dynamic Security Policy Support

With a *dynamic* network security policy [9] , the security restrictions and available security services depend on the network status or "mode" (e.g., normal, impacted, emergency, etc.) [8] .

Access to a predefined set of alternate security policies allows their functional requirements and implementation mechanisms to be examined with respect to the overall policy prior to being fielded, rather than depending on an ad hoc review. For example, during an emergency, a military commander might decide to forgo certain security protocols in order to get some important information transmitted quickly. This decision changes the security policy, but the actual policy arrived at may not be clearly understood.

If dynamic policies are created before deployment of the computer network, the network can respond to changing environments, while avoiding the confusion of ad hoc changes. A corporate intranet might have a mode indicating that the system is under attack from the internet. In this mode, it might be desired for a higher degree of network security to be in place. A military network might have an "emergency" mode indicating that there is a physical threat to the facility, and that command messages (only) which would normally be encrypted and signed, need to go out with the highest bandwidth available, disregarding cryptographic security. An ISP might have an "impacted" mode in which certain optional user security services would be curtailed for efficiency. In each of these cases, the changes to the security mechanisms would be predefined and limited to meet the desired alternate security policy.

In Table 2 , some hypothetical network modes are included in the translation matrix from Table 1 , showing how the "user security level" mappings would change, per mode. The modes are: normal, impacted and attack, as described above.

4 Conclusion

A security translation matrix can be used to provide users with a simplified representation of application and system security. Such a matrix can be used to translate user interfaces to a wide range of mechanisms, independent of how the mechanisms are related or distributed in the network. This mechanism can be used to support both variant and dynamic security policies.

References

[1] --, Department of Defense Trusted Computer System Evaluation Criteria, Department of Defense, DoD 5200.28-STD, December 1985

[2] --, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, National Computer Security Center, NCSC-TG-005, Version-1, July 1987

[3] Aurrecoechea, C., Campbell, A., and Hauw, L. "A Survey of Quality of Service Architectures", Multimedia Systems Journal, Special Issue on QoS Architectures, 1996.

[4] Hensgen, D., Kidd, T., St. John, D., Schnaidt, M.C., Siegel, H. J., Braun, T. Maheswaran, M., Ali, S., Kim, J-K., Irvine, C. E., Levin, T., Freund, R., Kussow, M., Godfrey, M., Duman, A., Carff, P., Kidd, S., Prasanna, V. Bhat, P., and Alhusaini, A., "An Overview of the Management System for Heterogeneous Networks (MSHN)," Proceedings of the 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, Apr. 1999, pp 184-198.

[5] Irvine, C., and Levin, T., Toward a Taxonomy and Costing Method for Security Metrics, NPS Technical Report, Forthcoming

[6] Johnson, Dale and Thayer, E. Javier, Security and the composition of machines, In Proceedings of the Computer Security Foundations Workshop, pages 72-89, IEEE Computer Society Press, June 1988.

[7] Juneman, R. R., Novel Certificate Extension Attributes--Novel Security Attributes: Tutorial and Detailed Design. Version 0.998, Novell, Inc. 122 East 1700 St., Provo, UT, August 1997.

[8] Kim, Jong-Kook, Hensgen, D., Kidd, T., Siegel, H.J., St.John, D., Irvine, C., Levin, T.,

Prasanna, V., and Freund, R., Priorities, Deadlines, Versions, and Security in a Performance Measure Framework for Distributed Heterogeneous Networks, Technical Report, Forthcoming.

[9] Levin, T., and Irvine C., Quality of Security Service in a Resource Management System Benefit Function, NPS Technical Report, Forthcoming

[10] Nahrstedt, K., and Steinmetz, R., Resource management in networked multimedia systems. IEEE Computer, 28(5):52-65, May, 1995.

[11] Rushby, John, Composing trustworthy systems: A position paper, In Proceedings of the NATO RSG2 Working Conference on Composability, October 1991

[12] Sabata, B., Chatterjee, S., Davis, M., Sydir, J., Lawrence, T. "Taxonomy for QoS Specifications," Proceedings the Third International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97), February 5-7, 1997, Newport Beach, Ca., pages 100-107

[13] Schantz, R. E. "Quality of Service," to be published in "Encyclopedia o Distributed Computing," 1998.

[14] Schneck, P. A., and Schwan, K., "Dynamic Authentication for High-Performance Networked Applications," Georgia Institute of Technology College of Computing Technical Report, GIT-CC-98-08, 1998.

[15] Stern, D. F., On the Buzzword "Security Policy", Proceedings of 1991 IEEE Symposium on Security and Privacy, 1991, Oakland, Ca., pages 219-230.

[16] Wang, C. and Wulf, W. A, "A Framework for Security Measurement." Proceedings of the National Information Systems Security Conference, Baltimore, MD, pp. 522-533, Oct. 1997.

Toward a Taxonomy and Costing Method for Security Services

Cynthia Irvine
Naval Postgraduate School
Monterey, CA

Tim Levin
Anteon Corporation
Monterey, CA

Abstract

A wide range of security services may be available to applications in a heterogeneous computer network environment. Resource Management Systems (RMSs) responsible for assigning computing and network resources to tasks need to know the resource-utilization costs associated with the various network security services. In order to understand the range of security services an RMS needs to manage, a preliminary security service taxonomy is defined. The taxonomy is used as a framework for defining the costs associated with network security services.

1 Introduction

Several efforts are underway to develop middleware *resource management systems* (RMSs) that will logically combine a wide range of network resources to construct a "virtual" computational system [2] [5] [10]. Geographically distributed, heterogeneous resources are expected to be used to support applications with a wide range of computation needs. Large parallelized computations found in fields such as astrophysics [11], aerodynamics, meteorology, etc. will require allocation of perhaps hundreds of individual processes. Multimedia applications, such as voice and video will impose requirements for low jitter, minimal packet losses, and isochronal data rates. Adaptive applications will need to adjust to changing conditions. The RMS in such an environment is responsible for: efficiently scheduling multiple simultaneous tasks onto specific network resources; supporting user requirements for performance and security (viz, QoS); and providing support for tasks to adapt to changing resource availability.

Users or applications submit tasks to the RMS, which schedules the tasks for execution. As part of the process of estimating efficient task schedules, the RMS must balance resource-usage costs against user benefits. Specifically, there might not exist sufficient resources to maximize the benefits to all users. Thus the RMS must quantify the costs associated with the entire range of network services. These costs include bandwidth, task execution speed, latency, storage, etc. Costing of security services in this context has received little attention. The challenge is to associate costs with the entire range of network security services.

The purpose of this paper is to present a preliminary taxonomy of security services, and to show how this taxonomy can be used as the foundation of a system for supplying security-costing information to an RMS. Section 2 presents our preliminary taxonomy. Section 3 is a sketch for how the structure of the taxonomy might be used to define quality of security service requests to an RMS. Section 4 examines how the cost of using various elements of the taxonomy might be presented to an RMS; and Section 5 is a discussion and conclusion.

2 Taxonomy of security services

Users and applications on the network are presented with various security *services* (e.g., authenticity, confidentiality, integrity, non-repudiation, etc.). A security service may be used to implement one or more security policies (organizational and automated [16]), and is, in turn, implemented by one or more security mechanisms (and of course, a given security mechanism may be used to implement different security services, e.g., "OS access controls," in Table 1). Some mechanisms provide fixed services, and some are variant.[1] Additionally, the RMS may make choices for the user regarding variant security mechanisms, as part of its schedule formulation or adaptive re-scheduling (see Section 4).

Each security mechanism is associated with a service area, which indicates the general topographical component of the network in which the security or protection is effective. The taxonomy identifies three service areas: end system (e.g., a client or server system), intermediate node (e.g., routers, switches), and network connection (i.e., the "wire" connecting various systems and nodes). Security

[1] Variant mechanisms offer the user various "degrees," or strengths, of security (viz., over and above some minimum requirement). See [9] for details.

To appear in *Proceedings of the Computer Security Applications Conference,* Phoenix, AZ, December 1999.

mechanisms associated with end systems and intermediate nodes protect resources (e.g., data and programs) that are associated with a node or system; for network connections, we are concerned with mechanisms for protecting information that is physically in transit.

Table 1 provides our preliminary taxonomy. It lists security services, example mechanisms and associated service areas. The service areas are designated: "IN" for Intermediate Node, "NC" for network connection, and "ES" for End System. The Total Subnet (TS) service area identifies mechanisms that cannot be assigned exclusively to either of IN, W, or ES.

2.1 Rationale for the taxonomy

In constructing a taxonomy one wishes it to be both useful and complete. Since a taxonomy is simply an organizational artifice, it must have reason to exist, which is its usefulness. Additionally, the taxonomy fails if it does not account for all of the elements of the classes that it attempts to organize.

We have found this taxonomy to be a useful tool for characterizing the security services and requirements that a RMS might encounter in the network context. As such, it is useful for organizing a quality of security service request (see Section 3) and for presenting costs to a Resource Management System (See Section 4).

As for completeness, we assert preliminarily that the top level is complete. Our taxonomy includes the traditional security categories found in the literature, e.g., Pfleeger [12] (confidentiality, integrity, availability), Ford [4] (authentication, access control, confidentiality, integrity, non-repudiation), and Stallings [15](confidentiality, integrity, availability, authentication, nonrepudiation, access control). (Note that in the latter two examples we find "access control" to be redundant with availability, confidentiality and integrity). Empirically, all of the example mechanisms that we have examined so far have been accounted for in our top level list of security services.

The second level (viz., end system, intermediate node, and network connection) is a simple enough partitioning of the generic network topology that we claim it to be complete through inspection. The list of mechanisms in Table 1 is not intended to be exhaustive, but provides a framework for illustrating the taxonomy.

3 Quality of security service requests

The security service taxonomy can be useful in understanding how security is involved in a Quality of Service request. Security in the Quality of Service context has traditionally implied the general notions of one or more of the following: confidentiality, authenticity, access control, and integrity [3] [13]. However, there is no reason why a Quality of Security Service request could not include all of the elements from "Security Service" and "Service Area" in Table 1. In other words, we envision a security vector in a fully-functional Quality of Service request to include levels of service for the range of security services and mechanisms that we have identified, where "level of service" can indicate degrees of security with respect to assurance, mechanistic strength, administrative diligence, etc. Thus, a generic QoS request would look something like the following in a BNF-style notation:

QoS Request ::= task_specifier, security_vector,
performance_vector, other_factors

And a security vector would appear as follows:

security_vector ::= security_component [,
security_component]*

security_component ::= security_service, service_area,
level

security_service ::= <services from Table 1>

service_area ::= [ES | IN | NC]

level ::= <mechanism-dependent security-level indicator>

A component may be included in the security vector for each variant security mechanism, i.e., for each mechanism in the network environment that provides to the user a choice of security "level." For example, a partial security vector might look like this:

data confidentiality, NC, crypto-high (e.g., 128-bit keys),

authenticity, NC, medium (e.g., public-key signature),

nonrepudiation, ES, high-assurance (e.g., Common Criteria rating EAL7 [1])

Here, for the sake of exposition, the "level" of each security component is somewhat arbitrarily assigned. Establishment of nomenclature and metrics for these levels is the subject of ongoing investigations [7] [18]. Translation mechanisms [6] may be utilized in presenting a high-level Quality of Security Service interface to the user, while managing parameters (such as a suitable translation of "level") to the underlying detailed security mechanisms.

4 Costing of security services

To motivate the need for security costing information, a specific RMS scheduling mechanism is described. We will

Table 1: Preliminary security service taxonomy

SECURITY SERVICE	SERVICE AREA	EXAMPLE SECURITY MECHANISMS
Data Confidentiality	IN	OS access controls, Cryptographic credentials
	NC	40-bit DES, 128-bit Blowfish
	ES	OS access controls, Cryptographic credentials
Traffic Flow Confidentiality	IN	Active network nodes monitor traffic and inject dummy packets in response to certain triggering conditions.
	NC	Communications use a Virtual Private Network with encapsulated packets
	ES	Traffic padding up to a defined maximum is provided. Beyond that maximum, traffic flow confidentiality cannot be guaranteed.
Data Integrity	IN	OS access controls, Cryptographic credentials
	NC	Cryptographic chaining, integrity sequence numbers, and digital signatures
	ES	OS access controls, Cryptographic credentials
Authenticity	IN	Active network supports internode authentication based on digital signatures
	NC	Data origin authentication, i.e., IP address, digital signatures
	ES	OS identification and authentication mechanism; use of Digital Signature Standard; use of trusted certificate authority
Non-Repudiation	IN	Active network nodes report transactions to secure logging facility
	ES	Digital notary and non-repudiation services
Guarantee of Service, Availability	IN	Active network nodes reserve bandwidth for network administrative traffic. Priority-based scheduling for application traffic
	NC	Bandwidth reservation protocol
	ES	Time-slicing scheduler, FIFO scheduler with preemptive interrupts
Audit and Intrusion Detection	IN	Auditing of network control functions
	TS	Rule-based and profile-based network intrusion detection, intrusion correlation engine to identify intrusions across a group of subnets
Boundary Control	TS	firewall, proxy server, guard

show how this work requires detailed security costing information.

Resource management systems are responsible for efficiently scheduling multiple tasks onto computing and network resources in a distributed, heterogeneous computing environment. RMSs support Quality of Service by scheduling to meet user requirements for performance and security, and by providing support for tasks to adapt to changing network resource availability.

An RMS schedules tasks for execution in the network in response to requests from users or applications. The task may be submitted with a QoS "specification," which articulates the user's desired quality of service, including security services. An RMS currently under investigation, the Management System for Heterogeneous Networks [5], has as its primary goal determination of the best scheduling

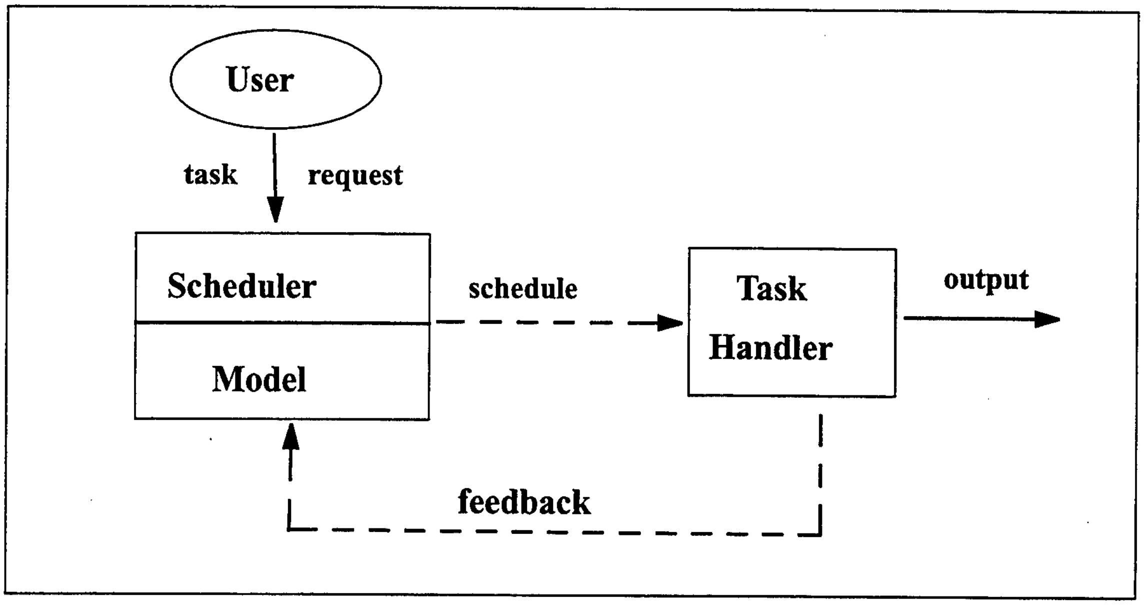


FIGURE 1. Resource Scheduler. The task handler is responsible for realizing the scheduler's execution plan and provides feedback to the scheduler so that it can dynamically adapt the schedule to evolving resource conditions.

support for many diverse applications, each with its own quality of service requirements, in a distributed, heterogeneous environment. MSHN preserves compatibility with existing security policies, applications and operating systems through its middle-ware role. This is in contrast to network operating systems, which *strictly* control the access to and utilization of resources, and usually require modifications to the OS, application, or security policy.

The MSHN RMS constructs task schedules based on a network infrastructure model. This model includes the resource and security requirements of current and waiting tasks, and the security and availability of network, computing and storage resources. The resulting schedules are provided to task handlers that run the tasks and provide feedback to the scheduler. If the model is inaccurate (e.g., security or resource availability changes), the RMS adjusts its model and potentially reschedules the tasks (see Figure 1).

RMS schedule construction consists of several logical phases, or steps:

1. In the reduction phase, the scheduler finds the *realizable* resource assignments for the task by discarding the possible assignments that will not work according to the model. In addition to resource availability matching (e.g., required service type vs. resource type), security plays a key role. Both the task and the resources are characterized by security requirements: those of the task must be met by a subset of the resources; those of the resources constrain the task. The task's security characteristics are compared to the minimum security requirements of the various resources and infrastructure components to determine where the task can run. Additionally, the task's minimum and maximum security requirements (e.g., reflecting the user's QoS security specification) are compared to the services available from the resources and infrastructure. The result is a set of resource-assignment "solutions," where each solution identifies various resources sufficient to run the task.
2. The resource usage costs, including costs for accessing security services, are derived for the various solutions.
3. In the optimization phase, an "optimum" solution is heuristically selected. The criteria for selection is to (attempt to) minimize costs and to maximize the QoS benefit to the users ([8] [9] [17]). More specifically, using realizable resources from the reduction phase, the scheduler attempts to create a schedule to meet QoS requirements for all of its tasks. In order to support as many tasks as possible, the scheduler must meet the typical task scheduling constraints while minimizing resource usage costs.

After step 3, some RMSs may make various network resource reservations. Finally, the task is submitted for execution.

If a particular security mechanism is "fixed" (i.e.,

Table 2: Security cost examples

Security Service	Service Area	Mechanism	Cost Measure
Data Confidentiality	NC	link layer 40-bit DES	Processor clocks per byte [14]
Message Non-Repudiation	ES	remote non-repudiation service	*2n* bytes per message network bandwidth, plus *c* clocks per byte
Intrusion Detection	TS	experimental ID system	*n* Mbytes per second of overall bandwidth, plus *m* instructions per second, plus *b* bytes per second storage

always applied) then the overhead for the mechanism is part of the normal cost of running the task and the normal costing mechanism used by the RMS will suffice. For variant security mechanisms, however, the security overhead will vary, depending on the user's QoS request. For example, some task invocations will utilize little, if any, of the variant mechanism; other invocations may utilize the mechanism at an increased level; and, the scheduler may adapt security support (while maintaining any minimum system security policy requirements) in order to schedule the tasks most efficiently. The RMS must calculate how much the use of the security mechanism will increase the cost of the task, according to the specific security "level" requested. For this reason, the RMS must have access to detailed information about the resource cost (as well as the task's requested QoS) for each variant security mechanism. Near-optimal solution selection depends on the accurate estimation of per-task, per-resource, cost of security.

With respect to implementation, the RMS's costing information may be table-driven or algorithm-based, and the cost measurement scale may vary for each mechanism and resource (see Section 4.1 , below).

4.1 Costing example

The security overhead for several security mechanisms is shown in Table 2.

The data confidentiality mechanism is a 40-bit DES encryption mechanism implemented in the link layer. For message non-repudiation, a commercial non-repudiation service mechanism is used. The cost of using this mechanism is a per-message exchange of n bytes with the remote non-repudiation server, and c clocks per message-byte to create the crypto-checksum for the message. The intrusion detection mechanism is shown to use a fixed overhead of the network bandwidth (e.g., for sampling and probing) along with constant processor and storage overhead

Costing information is provided to the scheduler, which will use these data and its current system model to select services, including those for security, that maximize the benefit for the collection of tasks it is serving [8].

5 Discussion and conclusion

A taxonomic framework has been presented for describing security services in terms of broad service categories, network "service areas," and security mechanisms. It has been shown that this taxonomy can be used for different purposes, including the specification of user Quality of Service requests, and the specification of security costs related to network tasks.With respect to Quality of Security Service, we have envisioned that users would be able to specify levels or ranges of desired security service, and as with other QoS parameters, could use these specifications to be able to trade off levels of task performance against requested levels of security.

Continued effort is required to determine the best units for the cost measures. For example, all measures could be unitless and normalized within a common framework. This approach would require a careful description of the semantics of the units with respect to each security service. Alternatively, units can be retained and the components combined into a "vector" to be used by the RMS scheduler.

Additionally, further work is required to expand the enumeration of specific security mechanisms with respect to the described taxonomy.

Acknowledgments

The authors wish to gratefully acknowledge partial support for this research from the DARPA/ITO program.

References

[1] Evaluation Criteria for IT Security, ISO/IEC CD 15408, Part 3: Security Assurance Requirements, November 1997. (commonly known as the *Common Criteria*)

[2] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115-128, 1997.

[3] Foster, I, Kesselman, C., Tsudik, G., and Tuecke, S., A

Security Architecture for Computational Grids, *Proceedings of the Fifth Conference on Computer and Communications Security*, San Francisco, CA, 1998, pp. 83--92.

[4] Ford, W., Computer Communications Security, Englewood Cliffs, NJ: PTR Prentice Hall, 1994, page 22.

[5] Hensgen, D., Kidd, T., St. John, D., Schnaidt, M.C., Siegel, H. J., Braun, T. Maheswaran, M., Ali, S., Kim, J-K., Irvine, C. E., Levin, T., Freund, R., Kussow, M., Godfrey, M., Duman, A., Carff, P., Kidd, S., Prasanna, V. Bhat, P., and Alhusaini, A., "An Overview of the Management System for Heterogeneous Networks (MSHN)," *Proceedings of the 8th Workshop on Heterogeneous Computing Systems (HCW '99)*, San Juan, Puerto Rico, Apr. 1999, pp 184-198.

[6] Irvine, C., and Levin, T., A Note on Mapping User-Oriented Security Policies to Complex Mechanisms and Services, Naval Postgraduate School Technical Report, NPS-CS-99-008, June 1999.

[7] Juneman, R. R., Novell Certificate Extension Attributes--Novel Security Attributes: Tutorial and Detailed Design. Version 0.998, Novell, Inc. 122 East 1700 St., Provo, UT, August 1997.

[8] Kim, Jong-Kook, Hensgen, D., Kidd, T., Siegel, H.J., St.John, D., Irvine, C., Levin, T., Porter, N.W., Prasanna, V., and Freund, R., A QoS Performance Measure Framework for Distributed Heterogeneous Networks, to appear in *Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing*, Rhodos, Greece, January 2000.

[9] Levin, T., and Irvine C., Quality of Security Service in a Resource Management System Benefit Function, NPS Technical Report, Forthcoming

[10] Litzkow, M. Livney, M., and Murtka, M. Condor: *A Hunter of Idle Workstations. In Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, CA, June 1998, pp 104-111.

[11] Ostriker, J., and Norman. M. L., *Cosmology of the Early Universe Viewed Through the New Infrastructure.* C.A.C.M. 40(11):85-94.

[12] Pfleeger, C., Security in Computing, Prentice Hall PTR, Upper Saddle River, NJ, 1997, page 4.

[13] Sabata, B. Chatterjee, S., Davis, M., Sydir, J., and Lawrence, T., "Taxonomy for QoS Specifications," *Proceedings of the IEEE Computer Society 3rd International Workshop on Object-oriented Real-time Dependable Systems (WORDS '97)*, Newport Beach, CA, Feb 1997.

[14] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Twofish: A 128-Bit Block Cipher, Counterpane Systems, http://www.counterpane.com/twofish-paper.html, June, 1998.

[15] Stallings, W., Network and Internetwork Security, Prentice Hall, Englewood Cliffs, NJ, 1995, page 5.

[16] Stern, D. F., On the Buzzword "Security Policy", Proceedings of *1991 IEEE Symposium on Security and Privacy*, Oakland, Ca., May 1991, pp. 219-230.

[17] Vendatasubramanian, N. and Nahrstedt, K., "An Integrated Metric for Video QoS." *ACM International Multimedia Conference*, Seattle, Wa., Nov. 1997.

[18] Wang, C., and Wulf, W.A., Towards a Framework for Security Measurement, *Proceedings of the Twentieth National Information Systems Security Conference*, Baltimore, MD, October 1997, pp. 522-533.

Toward a Taxonomy and Costing Method for Security Services[1]

Cynthia Irvine
Naval Postgraduate School
Monterey, CA

Tim Levin
Anteon Corporation
Monterey, CA

Abstract. *A wide range of security services may be available to applications in a heterogeneous computer network environment. Resource Management Systems (RMSs) responsible for assigning computing and network resources to tasks need to know the resource-utilization costs associated with the various network security services. In order to understand the range of security services an RMS needs to manage, a preliminary security service taxonomy is defined. The taxonomy is used a s framework for a preliminary method for defining the costs associated with network security services.*

1 Introduction

Several efforts are underway to develop middleware *resource management systems* (RMSs) that will logically combine a wide range of network resources to construct a "virtual" computational system [2] [5] [10]. Geographically distributed, heterogeneous resources are expected to be used to support applications with a wide range of computation needs. Large parallelized computations found in fields such as astrophysics [11], aerodynamics, meteorology, etc. will require allocation of perhaps hundreds of individual processes. Multimedia applications, such as voice and video will impose requirements for low jitter, minimal packet losses, and isochronal data rates. Adaptive applications will need to adjust to changing conditions. The RMS in such an environment is responsible for: efficiently scheduling multiple simultaneous tasks onto specific network resources; supporting user requirements for performance and security (viz, QoS); and providing support for tasks to adapt to changing resource availability.

Users or applications submit tasks to the RMS, which schedules the tasks for execution. As part of the process of estimating efficient task schedules, the RMS must balance resource-usage costs against user benefits. Specifically, there might not exist sufficient resources to maximize the benefits to all users. Thus the RMS must quantify the costs associated with the entire range of network services. These include bandwidth, task execution speed, latency, jitter, etc. Costing of security services in this context has received little attention. The challenge is to associate costs with the entire range of network security services.

The purpose of this paper is to present a preliminary taxonomy of security services, and to show how this taxonomy can be used as the foundation of a system for supplying security-costing information to an RMS. Section 2 presents our preliminary taxonomy. Section 3 is a sketch for how the

[1] This research was supported, in part, by the DARPA/ITO Quorum Program.

structure of the taxonomy might be used to define quality of security service requests to an RMS. Section 4 examines how the cost of using various elements of the taxonomy might be presented to an RMS; and Section 4 is a summary conclusion.

2 Taxonomy of Security Services

Users and applications on the network are presented with various security *services* (e.g., authenticity, confidentiality, integrity, non-repudiation, etc.). A security service may be used to implement one or more security policies (organizational or automated [16]), which are in turn implemented by one or more security mechanisms. Some mechanisms provide fixed services, and some are variant.[1] Additionally, the RMS may make choices for the user regarding variant security mechanisms, as part of its schedule formulation or adaptive re-scheduling (see Section 4).

Each security mechanism is associated with a service area, which indicates the general topographical component of the network in which the security or protection is effective. The taxonomy identifies three service areas: end system (e.g., a client or server system), intermediate node (e.g., routers, switches), and network connection (i.e., the "wire" connecting various systems and nodes). Security mechanisms associated with end systems and intermediate nodes protect resources (e.g., data and programs) that are associated with a node or system; for network connections, we are concerned with mechanisms for protecting information that is physically in transit.

Table 1 provides our preliminary taxonomy. It lists security services, example mechanisms and associated service areas. The service areas are designated: "IM" for Intermediate Node, "W" for wire, and "ES" for End System. The Total Subnet (TS) service area identifies mechanisms that cannot be assigned exclusively to either of IN, W, or ES.

2.1 Rationale for the Taxonomy

In constructing a taxonomy one wishes it to be both useful and complete. Since a taxonomy is simply an organizational artifice, it must have reason to exist, which is its usefulness. Additionally, the taxonomy fails if it does not account for all of the elements of the classes that it attempts to organize.

We have found this taxonomy to be a useful tool for characterizing the security services and requirements that a RMS might encounter in the network context. As such, it is useful for organizing a quality of security service request (see Section 3) and for presenting costs to a Resource Management System (See Section 4).

As for completeness, we assert preliminarily that the top level is complete. Our taxonomy includes the traditional security categories found in the literature, e.g., Pfleeger [12] (confidentiality/integrity/availability), Ford [4] (authentication/access control/confidentiality/integrity/non-repudiation) Stallings [15](confidentiality/integrity/availability/authentication/nonrepudiation/access control) (Note that in the latter two examples we find "access control" to be redundant with availability, confidentiality and integrity). Empirically, all of the example mechanisms that we have examined so far have been accounted for in our top level list of security services.

[1] Variant mechanisms offer the user various "degrees," or strengths, of security (viz., over and above some minimum requirement). See [9] for details.

Table 1: Preliminary Security Service Taxonomy

SECURITY SERVICE	SERVICE AREA	EXAMPLE SECURITY MECHANISMS
Data Confidentiality	IN	OS access controls, Cryptographic credentials
	W	40-bit DES, 128-bit Blowfish
	ES	OS access controls, Cryptographic credentials
Traffic Flow Confidentiality	IN	Active network nodes monitor traffic and inject dummy packets in response to certain triggering conditions.
	W	communications uses a Virtual Private Network with encapsulated packets
	ES	Traffic padding up to a defined maximum is provided. Beyond that maximum, traffic flow confidentiality cannot be guaranteed
Data Integrity	IN	OS access controls, Cryptographic credentials
	W	cryptographic chaining, integrity sequence numbers, and digital signatures
	ES	OS access controls, Cryptographic credentials
Authenticity	IN	Active network supports internode authentication based on digital signatures.
	W	data origin authentication, i.e. IP address, digital signatures
	ES	OS identification and authentication mechanism; use of Digital Signature Standard; use of trusted certificate authority
Non-Repudiation	IN	Active network nodes report transactions to secure logging facility.
	ES	digital notary and non-repudiation services
Guarantee of Service, Availability	IN	Active network nodes reserve bandwidth for network administrative traffic. Priority-based scheduling for application traffic.
	W	bandwidth reservation protocol.
	ES	time-slicing scheduler, FIFO scheduler with preemptive interrupts,
Audit and Intrusion Detection	IN	auditing of network control functions
	TS	rule-based and profile-based network intrusion detection, intrusion correlation engine to identify intrusions across a group of subnets
Boundary Control	TS	firewall, proxy server, guard

The second level (viz., end system, intermediate node, and network connection) is a simple enough partitioning of the generic network topology that we claim it to be complete through inspection. The list of mechanisms in Table 1 is not intended to be exhaustive, but provides a framework for illustrating the taxonomy.

3 Quality of Security Service Requests

The security service taxonomy may be useful in understanding how security is involved in a Quality of Service request. Security in the Quality of Service context has traditionally implied the general notions of one or more of the following: confidentiality, authenticity, access control, and integrity [3] [13]. However, there is no reason why a Quality of Security Service request could not include all of the elements from "Security Service" and "Service Area" in Table 1.[1] In other words, we envision a security vector in a fully-functional Quality of Service request to include levels of service for the range of security services and mechanisms that we have identified. Thus, a generic QoS request would look something like the following in a BNF-style notation:

QoS Request ::= task_specifier, security_vector, performance_vector, other_factors

And a security vector would appear as follows:

security_vector ::= security_component [, security_component]*
security_component := security_service, service_area, level
security_service ::= <services from Table 1>
service_area ::= [ES | IN | W]
level ::= <mechanism-dependent security-level indicator>

A component may be included in the security vector for each variant security mechanism, i.e., for each mechanism in the network environment that provides to the user a choice of security "level." For example, a partial security vector might look like this:

data confidentiality, W, crypto-high (e.g., 128-bit keys),
authenticity, W, medium (e.g., public-key signature),
nonrepudiation, ES, high-assurance (e.g., Common Criteria rating EAL7 [1])

Here, for the sake of exposition, the "level" of each security component is somewhat arbitrarily assigned. Establishment of nomenclature and metrics for these levels is the subject of ongoing investigations [7] [18]. Translation mechanisms [6] may be utilized in presenting a high-level Quality of Security Service interface to the user, while managing parameters (such as a suitable translation of "level") to the underlying detailed security mechanisms.

4 Costing of Security Services

To motivate the need for security costing information, a specific RMS scheduling mechanism is described. We will show how this work requires detailed security costing information.

Resource management systems are responsible for efficiently scheduling multiple tasks onto computing and network resources in a distributed, heterogeneous computing environment. RMSs support Quality of Service by scheduling to meet user requirements for performance and security, and by providing support for tasks to adapt to changing network resource availability.

An RMS schedules tasks for execution in the network in response to requests from users or appli-

[1] Given that the over-arching network security policy demands some minimum levels of security service, selections for QoSS may be provided to users to any degree of security over and above those minimum levels. A system can always provide more security, at the user's discretion, than the minimum required by the base security policy, while still complying with that policy. Finally, in order to meet performance or other objectives, a user may indicate a maximum security service to be provided by the system.

cations. The task may be submitted with a QoS "specification," which articulates the user's desired quality of service, including security services. An RMS currently under investigation, the Management System for Heterogeneous Networks [5], has as its primary goal determination of the best scheduling support for many diverse applications, each with its own quality of service requirements, in a distributed, heterogeneous environment. MSHN preserves compatibility with existing security policies, applications and operating systems through its middle-ware role. This is in contrast to network operating systems, which *strictly* control the access to and utilization of resources, and usually require modifications to the OS, application, or security policy.

The MSHN RMS constructs task schedules based on a network infrastructure model. This model includes the resource and security requirements of current and waiting tasks, and the security and availability of network, computing and storage resources. The resulting schedules are provided to task handlers that run the tasks and provide feedback to the scheduler. If the model is inaccurate (e.g., security or resource availability changes), the RMS adjusts its model and potentially reschedules the tasks (see Figure 1 on page 5).

RMS schedule construction consists of several logical phases, or steps:

1. In the reduction phase, the scheduler finds the *realizable* resource assignments for the task by discarding the possible assignments that will not work according to the model. In addition to resource availability matching (e.g., required service type vs. resource type), security plays a key role. Both the task and the resources are characterized by security requirements. Those of the task must be met by a subset of the resources. Those of the resources constrain the task. The task's security characteristics are compared to the minimum security requirements of the various resources and infrastructure components to determine where the task can run. Additionally, the task's minimum and maximum security requirements (e.g., reflecting the user's QoS security specification) are compared to the services available from the resources and infrastructure. The result is a set of resource-assignment "solutions," where each solution identifies various resources sufficient to run the task.
2. The resource usage costs, including costs for accessing security services, are derived for the various solutions.
3. In the optimization phase, an "optimum" solution is heuristically selected. The criteria for selection is to (attempt to) minimize costs and to maximize the QoS benefit to the users ([7] [9] [17]). I.e., using realizable resources from the reduction phase, the scheduler attempts to create a schedule to meet QoS requirements for all of its tasks. In order to support as many tasks as possible, the scheduler must meet the typical task scheduling constraints while minimizing resource usage costs.

After step 3, some RMSs may make various network resource reservations. Finally, the task is submitted for execution.

If a particular security mechanism is "fixed" (i.e., always applied) then the overhead for the mechanism is part of the normal cost of running the task and the normal costing mechanism used by the RMS will suffice. For variant security mechanisms, however, the security overhead will vary, depending on the user's QoS request. Some task invocations will utilize little, if any, of the variant mechanism and other invocations may utilize the mechanism at an increased level. Also, the scheduler may adapt security support, while maintaining any minimum system security policy requirements, in order to schedule the tasks most efficiently. The RMS must calculate how much

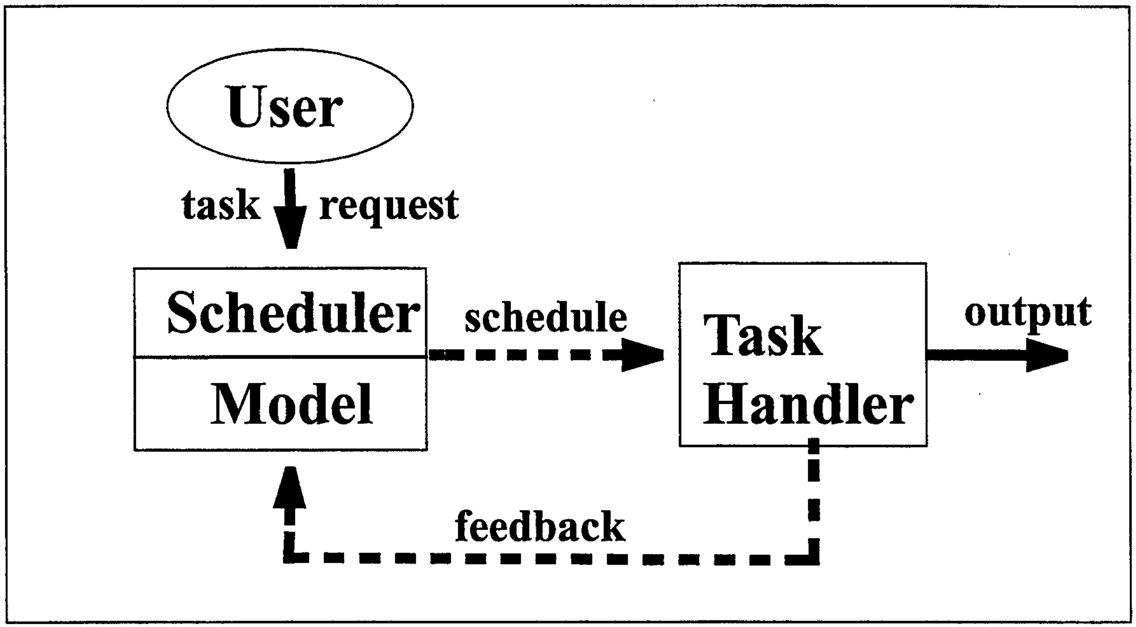


FIGURE 1. Resource Scheduler. the task handler is responsible for realizing the scheduler's execution plan and provides feedback to the scheduler so that it can dynamically adapt the schedule to evolving resource conditions.

the use of the security mechanism will increase the cost of the task, according to the specific security "level" requested. For this reason, the RMS must have access to detailed information about the resource cost (as well as the task's requested QoS) for each variant security mechanism. Near-optimal solution selection depends on the accurate estimation of per-task, per-resource, cost of security.

The RMS's costing information may be table-driven or algorithm-based. The cost measurement scale may vary for each mechanism and resource. A costing example follows.

4.1 Costing Example

The security overhead for several security mechanisms is shown in Table 2.

The data confidentiality mechanism is a 40-bit DES encryption mechanism implemented in the link layer. For message non-repudiation, a commercial non-repudiation service mechanism is used. The cost of using this mechanism is a per-message exchange of n bytes with the remote non-repudiation server, and c clocks per message-byte to create the crypto-checksum for the message. The intrusion detection mechanism is shown to use a fixed overhead of the network bandwidth (e.g., for sampling and probing) along with constant processor and storage overhead.

Table 2: Security Cost Examples

Security Service	Service Area	Mechanism	Cost Measure
Data Confidentiality	Wire	link layer 40-bit DES	Processor clocks per byte [14]
Message Non-Repudiation	ES	remote non-repudiation service	2n bytes per message network bandwidth, plus c clocks per byte
Intrusion Detection	TS	experimental ID system	n Mbytes per second of overall bandwidth, plus m instructions per second, plus b bytes per second storage

Costing information is provided to the scheduler, which will use these data and its current system model to select services, including those for security, that maximize the benefit for the collection of tasks it is serving [8].

5 Discussion and Conclusion

A security taxonomy has been presented for describing functional requirements of network security policies. It has been shown that this taxonomy can be used for different purposes, including a costing framework for network security mechanisms. Continued effort is required to determine the best units for the cost measures. For example, all measure could be unitless and normalized within a common framework. This approach would require a careful description of the semantics of the units with respect to each security service. Alternatively, units can be retained and the components combined into a "vector" to be used by the RMS scheduler.

References

[1] Common Criteria Project Sponsoring Organisations, *Common Criteria*, Version 2.0, Part 3: Security Assurance Requirements, CCIB 98-028, May 1998.

[2] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115-128, 1997.

[3] Foster, I, Kesselman, C., Tsudik, G., and Tuecke, S., A Security Architecture for Computational Grids, Proceedings of the Fifth Conference on Computer and Communications Security, San Francisco, CA, 1998, pp. 83--92.

[4] Ford, W., Computer Communications Security, Englewood Cliffs, NJ: PTR Prentice Hall, 1994, page 22.

[5] Hensgen, D., Kidd, T., St. John, D., Schnaidt, M.C., Siegel, H. J., Braun, T. Maheswaran, M., Ali, S., Kim, J-K., Irvine, C. E., Levin, T., Freund, R., Kussow, M., Godfrey, M., Duman, A., Carff, P., Kidd, S., Prasanna, V. Bhat, P., and Alhusaini, A., "An Overview of the Management System for Heterogeneous Networks (MSHN)," Proceedings of the *8th Workshop on Heterogeneous Computing Systems (HCW '99)*, San Juan, Puerto Rico, Apr. 1999, pp 184-198.

[6] Irvine, C., and Levin, T., A Note on Mapping User-Oriented Security Policies to Complex Mechanisms and Services, Naval Postgraduate School Technical Report, Forthcoming

[7] Juneman, R. R., Novel Certificate Extension Attributes--Novel Security Attributes: Tutorial and Detailed Design. Version 0.998, Novell,Inc. 122 East 1700 St., Provo, UT, August 1997.

[8] Kim, Jong-Kook, Hensgen, D., Kidd, T., Siegel, H.J., St.John, D., Irvine, C., Levin, T., Prasanna, V., and Freund, R., Priorities, Versions, and Security in a Performance Measure Framework for Distributed Heterogeneous Networks, *8th IEEE International Symposium on High Performance Distributed Computing*, Naval Postgraduate School Technical Report, Forthcoming.

[9] Levin, T., and Irvine C., Quality of Security Service in a Resource Management System Benefit Function, NPS Technical Report, Forthcoming

[10] Litzkow, M. Livney, M., and Murtka, M. Condor: *A Hunter of Idle Workstations*. In Proceedings of the 8th International Conference on Distributed Computing Systems

[11] Ostriker, J., and Norman. M. L., *Cosmology of the Early Universe Viewed Through the New Infrastructure*. C.A.C.M. 40(11):85-94.

[12] Pfleeger, C., Security in Computing, Prentice Hall PTR, Upper Saddle River, NJ, 1997, page 4.

[13] Sabata, B. Chatterjee, S., Davis, M., Sydir, J., and Lawrence, T., "Taxonomy for QoS Specifications," In the Proceedings of the IEEE Computer Society 3rd International Workshop on Object-oriented Real-time Dependable Systems (WORDS '97), Newport Beach, CA, Feb 1997.

[14] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Twofish: A 128-Bit Block Cipher, Counterpane Systems, http://www.counterpane.com/twofish-paper.html, June, 1998.

[15] Stallings, W., Network and Internetwork Security, Prentice Hall, Englewood Cliffs, NJ, 1995, page 5.

[16] Stern, D. F., On the Buzzword "Security Policy", Proceedings of *1991 IEEE Symposium on Security and Privacy*, Oakland, Ca., May 1991, pp. 219-230.

[17] Vendatasubramanian, N. and Nahrstedt, K., "An Integrated Metric for Video QoS." *ACM International Multimedia Conference*, Seattle, Wa., Nov. 1997.

[18] Wang, C., and Wulf, W.A., Towards a Framework for Security Measurement, Proceedings of the Twentieth National Information Systems Security Conference, Baltimore, MD, October 1997, pp. 522-533.

Quality of Security Service: An Introduction[1]

Cynthia Irvine
Naval Postgraduate School
Monterey, CA

Tim Levin
Anteon Corporation
Monterey, CA

Abstract. We examine the concept of security as a dimension of Quality of Service in distributed systems. We provide a discussion and examples of user-specified security variables and show how the range of service levels associated with these variables can support the provision of Quality of Security Service. We also discuss various design implications regarding security ranges provided in a QoS-aware distributed system.

Keywords. Quality of Service, Quality of Security Service, variant security, security range.

1 Introduction

Quality of Service (QoS) mechanisms benefit both the user and the overall distributed system. QoS users benefit by having reliable access to services; and the distributed systems whose resources are QoS managed benefit by having more predictable resource utilization and more efficient resource allocation (that is, in systems where allocation efficiency is supported). The motivation for the work described here has been to help determine if this reliability, predictability and efficiency can be enhanced by including security as a real part of QoS. We have termed the effects of this inclusion, "Quality of Security Service" (QoSS).

Inherently, QoS involves user requests for (levels of) services which are related to performance-sensitive variables in an underlying distributed system. For security to be a real part of QoS, then, security choices must be presented to users, and the QoS mechanism must be able to modulate related variables to provide predictable security service levels to those users. This raises the question of whether it makes sense within the context of coherent system security paradigms to provide such security choices to users. It is also of interest to understand how the limits on these choices are defined, and how those limits relate to existing resource security policies.

The purpose of this paper is to provide an overview, rationale and motivation for understanding QoSS and variant security, and how these concepts may benefit future application and system designs. The remainder of this document is structured as follows:

- Section 2 provides background on Quality of Service concepts related to security services;
- Section 3 describes the concept of Quality of Security Service, and provides a discussion of the general "assurability" of application-centric security enforcement mechanisms;
- Section 4 provides a description and rationale for various forms of user and application security "ranges;"
- Section 5 describes some design considerations regarding variant security in distributed multi-tiered systems; and
- Section 6 is a summary discussion.

1.1 Related Work

A *Quality of Protection* parameter is provided in the GSS-API specification [13]. This parameter is intended to manage the level of protection provided to a message communication stream by an underlying security mechanism

[1] This work was supported under the MSHN Project of the DARPA/ITO Quorum Program.

(or service), "allowing callers to trade off security processing overhead dynamically against the protection requirements for particular messages." Another early reference to a variable security service is that of Schneck, and Schwan [17], which discusses variable packet authentication rates with respect to the management of system performance. Our work is intended to extend these efforts into a more general framework which is applicable to a wide range of policy, processing and networking contexts, as well as diverse security services.

References to security in the QoS literature can be found in [6], [16], and [21], although little is mentioned there of security variability or use of security as a functional QoS dimension. QoS itself has been extensively discussed in the literature, and we refer the reader to [2] for a thorough review of QoS definitions and architectures.

A *trust management system* [3][4] provides a language and mechanism for specifying security policies and credentials, and may include a *policy server* or compliance checker to resolve questions about access control. The trust management system is not concerned with the nature of the specific policies (e.g., those involving variant security) which it stores and resolves. Nor is the trust management system expressly concerned with QoS issues. However, a QoSS system could be built to utilize a trust management system to store and resolve security range relationships.

2 QoS and Resource Usage Control

The resource usage load on traditional (e.g., not inter-networked) multi-user systems could be understood, simplistically, to be a linear function of the number of users. Similarly, user load could be seen as a function of the number of user terminals configured for the system. Thus, a system administrator could govern the system resource usage load, to a degree, by controlling the number and type of user input terminals (e.g., interactive terminals, modems and card readers). In a distributed and inter-networked environment, system administrators are often without recourse to such straightforward and simplistic resource-usage control approaches, since the number and type of user "terminals" and associated tasks may not be bounded by local (e.g., campus or enterprise) topographies. In some cases, users of system resources may extend across the Internet. The Quality of Service paradigm is designed to help address this problem by providing to users and administrators certain tools for managing resource usage and service levels.

Quality of Service refers to the ability of a distributed system to provide network and computation services such that each user's expectations for timeliness and performance quality are met. There are several dimensions of Quality of Service described in the literature [6][20], including, accuracy, precision and performance. For a Quality of Service *dimension* to be supported means that users can request or specify a level of service for one or more attributes of these dimensions, and the underlying QoS control mechanism is capable of entering into an agreement to deliver those services at the requested levels. Therefore, the control mechanism must be able to modulate the level of the service to individual subscribers (e.g., users). For example, a network-based multimedia application might be expected to deliver video frames so that the display is jitter-free to some requested level [20],[6].

In addition to meeting individual user requirements, a QoSM makes choices that permit it to maximize overall benefit in accordance with its QoS policy. For example, one QoS policy might require that benefit be equally shared among all tasks. This would mean that if network resources were over subscribed all tasks would have a reduction in service. Another policy might state that no service is better than poor service, so that if resources were sufficiently oversubscribed, some tasks would be postponed or terminated. This policy could be extended so that certain tasks would be given priority for guaranteed service during times of resource congestion.

Users present their expectations to the QoS mechanism by way of service level *requests*. These requests can take the form of both *hard* and *soft* requirements [19]. In essence, the system enters into a contract with the user to meet the hard and soft requirements. Hard requirements mandate fixed service levels that the QoS mechanism must deliver if it is to accept the user's task; whereas, a soft requirement can be considered to define a range of acceptable service, for example, in terms of bandwidth, response time, or image fidelity. Each soft requirement represents a variable which the QoS control mechanism can manipulate in balancing the needs of multiple users. Given latitude in the user's soft requirements, the more variables that the control mechanism has to manipulate, the easier will be the job of satisfying the set of current users. Conversely, the QoSM can offer choices to the user (in response to which the user may enter hard or soft requests) only for aspects of the system over which it controls, and is willing to provide, a range of service. For aspects in which there is no such control, only a fixed or "best effort" type of service can be delivered, so QoS concepts (e.g., regarding service level requests) do not apply.

3 Security

The purpose of this section is to provide an analysis of the role of security in a system designed to provide QoS. Security has long been a gleam in the eye of the QoS community: many QoS RFPs and QoS system-design presentation slides have included a place-holder for security, without defining security as a true QoS *dimension* (as above). Some of these presentations have provided access control mechanisms within the QoS framework [14][12], but they have only touched on security as a QoS dimension.

3.1 Quality of Security Service

We believe that QoS mechanisms can be more effective if, like response time and image fidelity, variable levels of security services and requirements can be presented to users or network tasks, providing security choices within acceptable ranges. As described above, these ranges result in additional tools (i.e., parameters) with which the QoSM can successfully meet overall user demands. Furthermore, if user security service requests are defined as ranges, then the underlying system can *adapt* more gracefully to changes in resource availability during the execution of a task, and thereby do a better job at maintaining requested or required levels of service in all of its dimensions. We use the term *Quality of Security Service* to refer to the use of *security* as a quality of service dimension.

To recap, the enabling technology for both QoSS and a security-adaptable infrastructure is *variant security*, or the ability of security mechanisms and services to allow the amount, kind or degree of security to vary, within predefined ranges. This notion of network Quality of Security Service has the potential to provide administrators and users with more flexibility and potentially better service, without compromise of network and system security policies.

3.2 Application-Centric Security

The traditional view of access control was OS-centric. The operating system enforced a policy, to the best of its ability, and ideally, objects never left the control domain of the OS. Policies that were enforced globally and persistently within this domain were considered to be "mandatory," and all others were considered to be "discretionary" [5]. With the advent of distributed/heterogeneous applications, data storage objects, operating systems, and resources, and a plethora of middleware mechanisms for managing those distributed entities, *application-centric access control* has now become common, if not the norm [3]. In this Brave New World, the application itself (perhaps in concert with some middleware mechanisms) enforces access control on its objects, rather than depending for this function on an underlying (e.g., OS and hardware) control mechanism. Thus, network applications have assumed some functions of the traditional OS. If the applications's objects are completely encapsulated, such that the object never leaves the control domain of the application[1], then a global and persistent policy could be said to be enforced, assuming persistence on the part of the application. However, this is a necessary, but not sufficient condition for effective policy enforcement.

Another traditional aspect of policy enforcement was the notion that, to be considered highly effective, access control should be performed at the lowest level(s), including hardware, of a strictly layered system. The reason for allocating access control functions to the lower levels is that it is more feasible, then, to ensure that the mechanisms are non-bypassable, persistently enforced, and small enough to allow thorough analysis (e.g., see [1]). Thus,

[1] Note that if the object is allowed to leave the application's domain, then it is more difficult to argue that a global policy is enforced; a component of one such argument for a distributed application is that objects in transit are protected, perhaps by cryptographic mechanisms, to the extent that the object remains, logically, in the control domain of the application.

regardless of how well formed or misused was an application, if the enforcement layers were well formed, the policy enforcement could be ensured. Modern distributed applications do not necessarily have these two properties (dependency layering, and access control implemented at the lowest levels). A network application typically depends on an untrusted operating system for access to resources, and we suggest that no application under these conditions can be considered to enforce a security policy with high effectivity (assurance). Neither is dependency layering a fundamental design consideration in many modern distributed or object-oriented applications and systems. As a result, the distributed application needs to be analyzed very carefully to understand whether or not it has the capability, by virtue of its design, to enforce a policy; for without understanding the dependency layering, it will not be clear on which other modules the application depends, nor will it be clear if there are fatal (e.g., circularly dependent) or semantically undefined execution sequences. Therefore, under the conditions described here, much more design analysis may be involved in understanding the degree to which a distributed system is capable of enforcing a security policy, than was required to analyze a traditional layered system.

We present in the following pages some thoughts about how certain QoS aspects of application-centric access control security can be understood and managed. This is not to say that this approach ameliorates the design analysis problems of application-centric access control. On the contrary, we would reiterate that each such system needs careful design review to understand the effectiveness of its security mechanisms. Hopefully, the security abstractions presented here will aid in such analyses.

4 Security Ranges

The notion of security ranges may, at first, seem strange or even an oxymoron. For many, security is thought to be binary: either you have it or you don't. On a gross scale, this is true. Without some minimum level of security, a system will be considered inadequate for user requirements. Yet if a user's minimum requirements are met, can there not be some choice with respect to what *is* adequate? Our answer is "yes." As an initial example, suppose that a user requires medium assurance at end systems where a distributed task will be executed. If potential target platforms range between medium and high assurance, there is a choice. In fact, if the medium assurance system is over-subscribed while the high assurance system is idle, the user may realize better overall service by electing to execute the task on the high assurance processor.

Consider the security administrator's or the user's motivation in agreeing to or specifying a range of security. As with multimedia image resolution, users will generally desire the greatest amount of security (or image fidelity) available, but this desire is generally tempered by cost. The cost may take the form of monetary charges (unlimited bandwidth but at a high cost per byte) or performance degradation (for high resolution, processing and download times will be long), for example. When the cost is very high (e.g., slow response time), users may be willing to accept security (or imagery) that is less than their ideal level of service. Thus, the user/administrator's *acceptable* security would range from a *minimum* to an *ideal*. A system that is sufficiently flexible may be able to impose performance degradations on others when an application that is willing to pay enough or has the highest priority is introduced. By indicating a range within which they are willing to operate, the poorer or lower priority tasks will still be able to run rather than being terminated or rejected.

Yet, once a user (or security officer) decides on the minimum level of security required for a given application, why would they ever agree to more security, if it increases their cost? For one, the level of security might be tied to another desirable service level, such as image fidelity. An application may have variable data formats which have correspondingly variable security requirements, as shown in Table 1. Here, the degraded image requires less security, and conversely, the enhanced image requires more security. So a user might welcome heightened security if it is tied to her desire for more image fidelity.

Table 1: Security Choice Related to Fidelity Choice

Fidelity	Security	Performance
high	high	low
medium	medium	medium
low	low	high

An example taken from a popular military novel will help to illustrate our point. Suppose that high, medium and low resolution images of enemy troop movements are available. Here we will assume that resolution and fidelity are equivalent. To protect the technologies used to obtain both the high and medium resolution images, correspondingly high and medium confidentiality mechanisms are required to protect the images, which will be handled as two levels of classified information. The images will be useful for military purposes only for a limited time, because within a few days the battle will be over. Those planning the battle strategy consider medium confidentiality to be sufficient to protect the images since even medium strength security mechanisms are considered sufficient to protect the information for a few days. However, to be useful for tactical planning high resolution is required. This means that the military strategists must employ high confidentiality mechanisms. Thus we have a situation in which the tactical field commander would be happy enough with medium security but her requirement for high resolution (or high fidelity) imposes a requirement for high security. The combined requirements for high fidelity and high security consume processing and network resources to produce low overall performance.

From the above example, we can also observe that if the fidelity of the images is diminished, the requirements for security are also reduced. If the images become fuzzy enough, little or no security is required. In this case, resource usage will be low and overall performance will be high.

An integrity example may also be useful. Suppose that a surgeon is performing a delicate brain operation remotely. To ensure that only the precise brain locations are affected, high fidelity is required. Additionally, there is a requirement for high integrity to ensure that the video stream is not tampered with by malicious entities who might wish to ruin the operation and render the patient a vegetable. Secrecy is not a requirement, yet to achieve the performance required, a hardware mechanism that provides both a high level of secrecy and integrity could be used. In this case the operation achieves high secrecy as a bonus resulting from fidelity and integrity requirements.

The following are some more examples of the use of real and hypothetical security ranges.

- Collaborative applications, such as video teleconferencing with shared electronic white boards, and application suites, may present communication security choices to participants. For example, if a group member is participating in the collaboration from a hotel room in a foreign country known for government support of corporate espionage, his security requirements and choices will be quite different than if he were in "friendly" territory. These security choices may form a range from which the user or application can select, and can include different levels of authentication, confidentiality, and integrity.

- Destination subnets could be classified by risk factor with respect to routing through, execution on, or logging on to nodes in those subnets. Users, applications or enterprise-wide mechanisms could request of middleware control mechanisms that communications or tasks executed on the user's behalf utilize a specific *risk range* of subnets (e.g., the user's QoSS specification might include the request to use any "high to very high" security subnets for this invocation).

- Some environments may offer the user choices of log-on authentication technology. For example, a user may log on with a password, a one-time password (crypto challenge-response), a public-key smart card, a biometric, or some combination of these. In these environments, the user could be granted greater access to resources (e.g., a higher classification of data) if he uses higher-assurance authentication [11].

- Another example is that the underlying system supports different *situational modes*. For some modes (e.g., normal, impacted, emergency), the user or administrator may be willing to accept more (or less) security for a given application. A commander under attack at a foreign embassy might require the highest communication security; whereas a commander under attack on the battlefield might declare, "damn the security, full speed ahead!" The MSHN resource management system is an example of a system in which the management of mode vs. security requirements is designed to be handled automatically [8][9].

- The security policy for a hypothetical commercial sub-network requires outgoing IP packet encryption. In this environment, a multimedia application exports digital images (e.g., high resolution fine art images). However, recognizing that the stake-holders in this specific environment can tolerate a media stream which is *partially* or *periodically* encrypted (viz, one yielding a suitably obscured image, which would render a stolen image unusable by the vast majority of its target market), the policy may only require that a range of from 80% to 100% of the packets should be encrypted. (Note that in some risk models, such a periodic encryption method might require fortified protection against cryptanalysis. In addition, care must be taken to ensure that the entire unencrypted image is not revealed in repeated transmissions.)
- Variable packet authentication [17] is a corollary to the preceding confidentiality scenario. In this case, the sender or recipient might be satisfied if (only) a certain percentage of the packets in an image stream were authenticated (e.g., 80% to 100%). Depending on the threat model and the packet-checking algorithm, to detect attacks attention may need to be paid to the ratio of good to bad packets: if all of the packets were bogus, and only 80% were checked, it might be possible for the display program to show a completely bogus image, with the remaining 20%.
- An administrator may choose to run an intrusion detection system within a range rather than at a fixed level. There would be a minimal level of IDS processing below which the system would not be permitted to fall, but the IDS would be balanced against performance requirements of the organization's tasks. Thus the IDS might perform more thoroughly (with deeper histories) when the system is lightly loaded than during peak hours. The administrator might also choose to set an upper limit to IDS performance.
- Another variable packet authentication scheme [22], would be to authenticate only a certain percentage of each packet; this might have applicability for image display, especially considering that the low order bits of each byte are not very significant, visually, in some display protocols.

The following are some example security variables, with characterizations of how they could be specified or measured:

- Strength of cryptographic algorithm, e.g., RSA, DES

  measured in terms of the work factor associated with a brute force attack
- Length of cryptographic key

  characterized by bit-length
- Security functions present in destination job-execution environment

  characterized by operating system or boundary control security policy enforcement mechanisms
- Confidence of policy-enforcement in remote login environment

  characterized by third-party evaluation
- Robustness of authentication mechanism

  here the range might span weak password, strong password, biometric, and smart cards with on-board display and input interfaces

From these examples, it is apparent that the notion of security ranges is useful and, in some cases, already evident in existing systems. Thus, we can conclude that it is reasonable to consider such ranges within the context of a QoS manager.

5 System Considerations

This section presents some observations about how variant security can be viewed in a distributed system which provides QoS support.

5.1 Security Resources, Services and Requirements

A network *system* is defined as the totality of network-accessible resources. A *security service* is a high-level abstract resource providing security functionality such as: authentication, auditing, privacy, integrity, intrusion detection, non-repudiation, and traffic flow confidentiality [10]. A security service typically consumes other low-level system resources such CPU, memory, disk, and network bandwidth. For example, the Common Data Security Architecture (CDSA) [15] describes modules, each of which contain specific security mechanisms to provide some of these services.

Each resource (including security services) may embody security *requirements* regarding its use. A requirement may restrict the availability of a resource to an external entity. Some restrictions might be the typical MAC and DAC requirements, or other security constraints, e.g.: encryption available 9 P.M. to 5 A.M., range of available encryption algorithms, and range of required key lengths.

To be general, we state that all security requirements define a *range* of permissible behavior. That is, a range may be *unitary*, or degenerate, in which case it represents no choice. Where a range represents a choice, the requirement is called *security variant.*

5.2 Task Sequences

Quality of service can be provided at several levels within the overall system. The notion of translucence, by which components can adapt to changing conditions at one or more other system or network layers, results in a problem that is both horizontal, viz. distributed across the network; and vertical, viz. distributed within the stack. In the following discussion, the management of QoSS can be seen to have both horizontal and vertical interactions, depending on the implementation of the various components.

A *task* is an application invoked by a user. The task utilizes various network system services and other resources. This utilization may be intermediated by different QoS middleware mechanisms (QoSMs), such as: QoS-aware object request brokers and application servers, distributed resource management systems, and various network traffic managers. In these multiple-tiered environments, a task is invoked in a *task invocation sequence*:

- the user activates the application through some interface with an application manager (OS, browser, etc.);
- the application is intermediated by the QoSM; and
- the QoSM submits the application to the system[1].

Security requirements may be established or refined by any or all of: the user, the application, the QoSM, and the system; we call these entities *security requirement providers*.

As an example of how a requirement can be refined within the task invocation sequence, consider how a typical application offers the user a choice for some service. If the user does not indicate a choice, the application may use a default value. If the user chooses a range, the application may invoke itself with a particular value within that range. Similarly, the QoSM may refine the application's choice, for example, to optimize the overall *system* (user population) performance, perform load balancing, etc.

[1] It is an implementation detail whether the QoSM returns advisory parameters to the application and the application invokes the system, or the QoSM submits the application with those parameters directly to the system. For simplicity, we assume, here, that the QoSM submits the application to the system.

5.3 Security Limits and Choices

In a task invocation sequence, the request is passed from a *previous* requirement provider to the *next* provider. A security choice for each variant security requirement is logically included with each request step. The choice may be implicit or explicit. For example, if no explicit choice is made, then it may be implicit that the choice is to not limit or modify the security options proffered at that step. As with requirements, all security choices define a *choice range*, which may be unitary. Thus, each requirement provider specifies a choice range for each variant requirement in a given task invocation. For example, the user selects a range of 50 - 80% for packet authentication rate. This choice is passed to the next provider (viz., the application) in the sequence.

For each variant security requirement, each requirement provider may also have an explicit requirement *limit range* (again, unitary or variant) outside of which it will not accept a request. The limit applies to the request choice from the previous provider, e.g., a given application will not accept a range wider than 60 - 100% from the user.

5.4 Security Range Relationships

Table 2 on page 7 shows the various limits and choices we have identified for security requirement providers in a task invocation sequence.

Table 2: Security Limits and Choices

	User	Application	Middleware	System
Choice Range provided	Yes	Yes	Yes	Service Level
Limit Range enforced	No	Yes	Yes	Yes

Notice that the user does not have an effective limit range, as he has no previous provider upon whom to enforce such a range. Also, the system choice range is the level of service ultimately provided by the system in response to the request. This is a unitary range, since there is no next provider to whom a choice might be given.

With so many requirement ranges at different points in the sequence, how do these ranges relate to each other? The following relationships appear to be inherent in a task invocation sequence:

1. The maximum of each limit and choice range *dominates*[1] the minimum of that range.

2. Each provider's choice range must be within its own limit range. This restriction reflects the natural protocol to respect one's own limits.

3. Each choice range must be within the previous choice range in the sequence. This reflects a natural protocol to respect the choice of the previous requirement provider: a requirement provider will try to fulfill the request of a previous provider. For example in a quality of service context, a service provider may accept a request if it can be realized, but it will not proceed with parameters which are divergent from (outside) the user's request.

4. Each choice range must be within the next limit range in the sequence. This restriction means that requests which are out of bounds will be rejected.

5. The limit ranges of each provider in a task sequence must all *intersect*. This is a consequence of the need for a choice to be within the provider's own limit, and within the next limit, as well as within the previous choice. Obviously, if two ranges in a task invocation sequence don't intersect, there does not exist a value which could

[1] For each variant security requirement there is a set of elements which are partially ordered by a "security" relation (dominates), and each range is a sub-lattice of that set such that the maximum of the range is more secure than the minimum. One range is contained "within" a second range, if and only if the max of the first dominates the max of the second, and the min of the second dominates the min of the first. For two ranges to intersect means that the maximum of each dominates the minimum of the other.

satisfy both ranges; this would disallow a task from execution.

These relationships are illustrated in Figure 1.

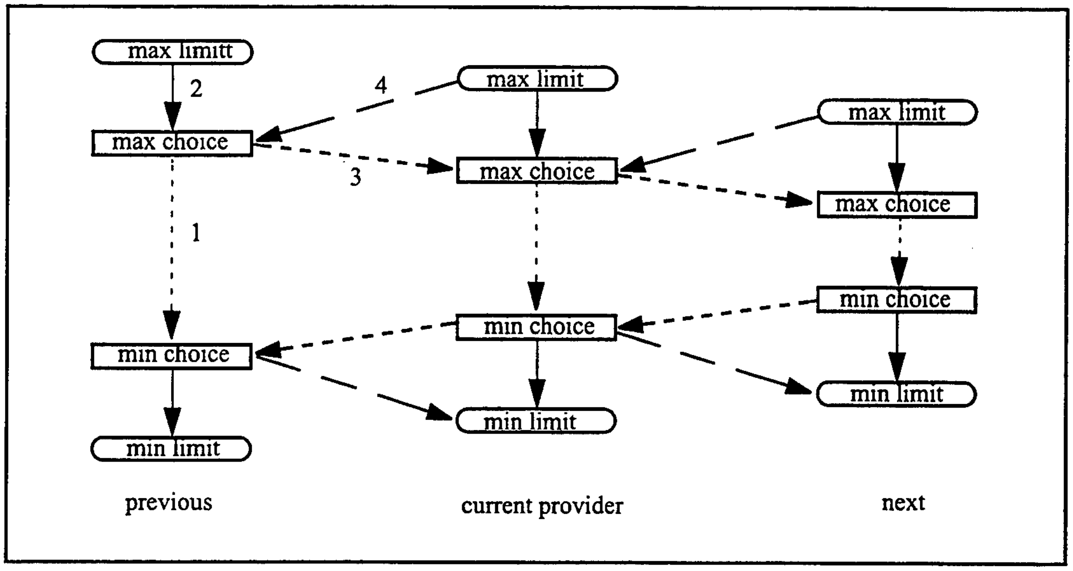


FIGURE 1. Relationships of Limits and Choices

Because the choices and limits are partially ordered and consequently comparable, it is possible for a security service selection algorithm to be encoded. A QoSM would maintain databases of static and dynamic resource characteristics. In the static database, limits might be recorded while the dynamic database could record current network conditions and choices. Thus when a new job enters the system, the QoSM can compute its execution strategy. We note that this is an NP-complete problem and extensive work exists on heuristic scheduling techniques, e.g. [18].

6 Summary

Our goal has been to provide an understanding of QoSS and variant security, and to determine whether these concepts can be useful in improving security service and system performance in QoS-aware distributed systems.

We described the general requirements for system attributes to participate in the provision of Quality of Service, and described how certain security attributes might meet these requirements. We then described various forms of user and application security "ranges" and showed how these ranges can make sense in relation to existing security policies, when those ranges are presented as user choices. Finally we described security ranges as forming a coherent system of relationships in a distributed multi-tiered system.

Our conclusion is that it may be possible for security to be a semantically meaningful dimension of Quality of Service without compromising existing security policies. Further study is needed to understand the effectiveness of QoSS in improving system performance in QoS-aware systems.

References

[1] Anderson, J.P., Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, 1972. (Also available as Vol. I, DITCAD-758206, Vol. II DITCAD-

772806).

[2] Aurrecoechea, C., Campbell, A., and Hauw, L. "A Survey of Quality of Service Architectures", Multimedia Systems Journal, Special Issue on QoS Architectures, 1996.

[3] Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A., The KeyNote Trust-Management System, version 2, RFC 2704, September 1999. ftp://ftp.ietf.org/internet-drafts/draft-blaze-itef-trustmgmt-keynote-02.txt

[4] Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A., The Role of Trust Management in Distributed System Security, to appear in Secure Internet Programming: Security Issues for Mobile and Distributed Objects, ed. Jan Vitek and Christian Jensen, Springer=Verlag Inc., New York, NY.

[5] Brinkley, D.L. and Schell, R. R., Concepts and Terminology for Computer Security, in Information Security: An Integrated Collection of Essays, ed. Abrams, Jajodia and Podell, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 40-97.

[6] Chatterjee, S., Sabata, B., Sydir, J. "ERDoS QOS Architecture," SRI Technical Report, ITAD-1667-TR-98-075, Menlo Park, CA, May 1998.

[7] Condell, M., Lynn, C. and Zao, J. "Security Policy Specification Language," INTERNET-DRAFT, Network Working Group, July 1, 1999, ftp://ftp.ietf.org/internet-drafts/draft-ietf-ipsec-spsl-01.txt, Expires January, 2000

[8] Hensgen,D., Kidd, T., St. John, D., Schnaidt, M., Siegel, H.J., Braun, T., Kim, J-K, Ali, S., Cynthia Irvine, Tim Levin, Prasanna, V., Bhat, P., Freund, R., and Gherrity, M., An Overview of the Management System for Heterogeneous Networks (MSHN), 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, Apr.1999

[9] Irvine, C., and Levin, T., A Note on Mapping User-Oriented Security Policies to Complex Machanisms and Services, NPS Technical Report, NPS-CS-99-008

[10] Irvine, C., and Levin, T., Toward a Taxonomy and Costing Method for Security Metrics, Annual Computer Security Applications Conference, Phoenix, AZ, Dec. 1999

[11] Juneman, R. R., Novell Certificate Extension Attributes--Novel Security Attributes: Tutorial and Detailed Design. Version 0.998, Novell, Inc. 122 East 1700 St., Provo, UT, August 1997.

[12] Lee, C. Kesselman, C., Stepanek, j., Lindell, R., Hwang, S., Scott Michel, B., Bannister, J., Foster, I., and Roy, A. The Quality of Service Component for the Globus Metacomputing System. Proc. 1998 International Workshop on Quality of Service, Napa California, pp. 140-142, May, 1998.

[13] Linn, J., Generic Security Service Application Program Interface, IETF Request for Comments: 1508, September 1993

[14] Sabata, B., Chatterjee, S., Davis, M., Sydir, J., Lawrence, T. "Taxonomy for QoS Specifications," Proceedings the Third International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97), February 5-7, 1997, Newport Beach, Ca., pages 100-107.

[15] Sargent, R., "CDSA Explained: An Indispensable Guide to Common Data Security Architecture",The Open Group, Reading, Berkshire, UK, 1998.

[16] Schantz, R. E. "Quality of Service," to be published in "Encyclopedia of Distributed Computing," 1998.

[17] Schneck, P. A., and Schwan, K., "Dynamic Authentication for High-Performance Networked Applications," Georgia Institute of Technology College of Computing Technical Report, GIT-CC-98-08, 1998.

[18] Siegel, H..J. and Ali, S., Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems, to appear in Journal of Systems Architecture, Special Issue on Heterogeneous Distributed and Parallel Architectures: Hardware, Software and Design Tools.

[19] Stankovic, J. A., M. Supri, M., Ramamritham, K., and Buttazo, G. C., "Deadline Scheduling for Real-time Systems, Kluwer Academic Publishers, Norwell MA, 1998, pp. 13-22.

[20] Vendatasubramanian, N. and Nahrstedt, K., "An Integrated Metric for Video QoS," ACM International Multimedia Conference, Seattle, Wa., Nov. 1997.

[21] Welch, L., Shirazi, B., and Ravindran, B., "DeSiDeRaTa: QoS Management Technology For Dynamic, Scalable, Dependable, Real-Time Systems," 15th Symposium on Distributed Computer Control Systems (DCCS'98), IFAC, Sept. 1998.

[22] Xie, G., Irvine, C., and Colwell, C., A Protocol for High Speed Packet Authentication, NPS Technical Report, NPS-CS-99-001, August 1999.

The Effects of Security Choices and Limits in a Metacomputing Environment

Cynthia E. Irvine
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943

Timothy Levin
Anteon Corp.
Monterey, CA 93940

Abstract

It is anticipated that the introduction of metacomputing and distributed resource management mechanisms to the Internet and World Wide Web will make available to users and applications a large diversity of previously unavailable network and computing resources. New methods of managing the scheduling and allocation of distributed resources bring into focus new problems and approaches for managing security in those contexts. We present an analysis of layered and variable security services and requirements. These services and requirements may be accessed via a network control program such as a Resource Management System (RMS) which is responsible for scheduling resources in distributed heterogeneous environments. The RMS will not present the same "virtual computer/network" to the same job each time it is submitted for execution. Each instance will be comprised of potentially different actual resources with different properties. Our objective is to understand how user and application requirements, characterized as choices and limits, can affect the overall security provided. A method is presented for fairly measuring the effectiveness of an RMS in performing security allocation and assignments with respect to security choices made by metacomputer users and applications. [1]

Keywords: Quality of Security Service, Resource Management System

1 Introduction: Managing Metacomputer Resource Allocation

Metacomputing provides users and applications with access to a virtual machine consisting of a wide range of distributed networking and computing resources (see e.g., [15]). Initially, efforts in metacomputing were focussed on providing transparent access to remote supercomputers for their user communities and support organizations. The advent of standardized protocols for (1) managing production and transmission of multimedia data [16], and (2) the more general distribution and execution of remote code (e.g. via the World Wide

[1] This work was supported under the MSHN Project of the DARPA/ITO Quorum Program

Web, Java, or Jini) may help to enable the vision of metacomputing to extend to devices and computational resources that are generally available on the Internet.

Whereas current distributed systems and internet technology may import mobile code for local execution (e.g., via Java applet), or request that remote code be executed in its native (fixed) environment (e.g., via servelets or object request brokers), metacomputing expands this paradigm to include execution of mobile code utilizing a wide range of possible remote resources. In some sense, recent Jini (Sun) and Universal Plug-and-Play (MS) technologies enable some metacomputing functions, but they may lack the ability to optimize multi-task scheduling or Quality of Service, or to adapt to changing resource availability.

The resources available on a metacomputing virtual machine are both local and remote; are implemented in hardware as well as software; and include processing, storage, and display devices. The heterogeneity [4], multiplicity and remoteness of these resources provides various management, scheduling and security challenges [5, 2]. Of specific concern for this paper is the fact that the metacomputer presents too many variables and choices for users or applications to manage without automated support.

Resource Management Systems (RMSs) are designed to provide efficient, automated management and allocation decisions for metacomputer resources [9, 8]. Allocation decisions involve matching requirements to capabilities and attributes for security [3], completion time, computational environment [12], network bandwidth, etc. The efficiency of an RMS in providing these type of decisions can be measured with respect to various user and system goals, for example, *quality of service* (QoS) specifications and system allocation policies [11, 16]. The relationship of the RMS to the metacomputer is shown in Figure 1, where P indicates compute or processor resources; N, network bandwidth; and D, data storage and data staging components.

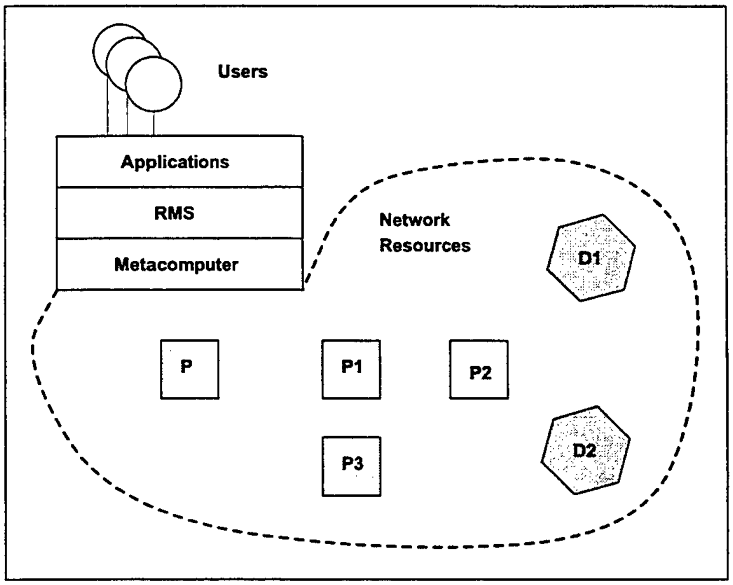


Figure 1: Metacomputing and the RMS

1.1 QoS Choices

Quality of Service refers to the ability of a system to provide services such that user expectations for timeliness and performance quality are met. For example, a multimedia application should deliver video frames so that the display is jitter-free [16, 6]. Quality of service can be provided at several levels within the overall system. The notion of translucence, where components can adapt to changing conditions at one or more other levels, results in a problem that is both horizontal, viz. distributed across the network; and vertical, viz. distributed within the stack. Finally quality of service requirements may change in systems supporting dynamic policies based upon current operating modes, e.g. normal, impacted or crisis [3].

Users have expectations with respect to the security services they are provided. These expectations may include both functional and assurance characteristics. With respect to security for a particular job, a user might require a minimum level of both functional mechanism and assurance. The ability of the network to meet these requirements is measured in terms of *Quality of Security Service* (QoSS) [3]. The notion of network Quality of Security Service expands the network service choices available through the metacomputer, providing administrators and users with more flexibility and potentially better service, without compromise of network and system security policies.

1.2 Security Ranges

As introduced in [3, 11], the security requirements presented to a network application can allow a *range* of security behavior. For example, a security policy for a hypothetical sub-network requires IP packet encryption. In this sub-net, a commercial multimedia application exports digital images (e.g., movies, or high-resolution fine art images). However, recognizing that the application in this specific environment can tolerate a media stream which is *periodically* encrypted (viz., one yielding a suitably obscured image, which would render a stolen image unsalable), the policy may only require that a range of from 80% to 100% of the packets should be encrypted. (Note that in some risk models, such a periodic encryption method might require fortified protection against cryptanalysis. In addition, care must be taken to ensure that in five repeated transmissions the entire unencrypted image is revealed.)

Collaborative applications, for which video teleconferencing with shared electronic white boards and application suites represent current technology, present another example in which security choices are available to the participants. Suppose that today one party in the group is located at organizational headquarters while another is a "road-warrior" participating from a hotel room in a foreign country known for government support of corporate espionage. Clearly the security requirements and choices of the road-warrior will be quite different than those chosen tomorrow when all participants will be in "friendly" territory. When a remote user is involved, collaborators may demand increased levels of both confidentiality and integrity support.

Consider the security administrator's or the user's motivation in agreeing to or specifying a range of security protection. As with multimedia image resolution, users will generally desire the greatest amount of security (or image fidelity) available, but this desire is tempered by cost. Cost may may take the form of monetary charges or performance degradation, for example. When cost is very high (e.g., slow image display), users may be willing to accept degraded security or imagry, instead.

Yet, once a user (or security officer) decides on the minimum level of security required for a given application, why would they ever agree to more security, if it increases their cost? For one, an application may have variable data formats, which may have correspondingly variable security requirements. A degraded image might require less security, and conversely, the enhanced image might be more security sensitive. To illustrate this example, a range of fidelity/security is shown in Table 1:

Table 1: **Security Ranges**

Fidelity	Security	Performance
high	high	low
medium	medium	medium
low	low	high

Another example is that the underlying system might support different situational modes. For some modes (e.g., "emergency"), the user or administrator may be willing to accept more (or less) security for a given application. The management of mode and security-level negotiation is handled automatically by some resource management systems [9].

Yet another scenario is that the underlying control program may have more flexibility to execute the job quickly, or at all, if the user can live with a range of security requirements. For example, transmission paths may go through a wide range of security environments. So the user specifies: do what you need to do, but give me at least "this much" security. The application might even execute faster with more security; regardless, the RMS manages the security allocation within the bounds specified by the user.

From the system's point of view, as opposed to that of the user, security variability provides another tradeoff factor, allowing the system to be more flexible in providing QoS for the system as a whole.

Here are some other examples of security ranges with examples of how the ranges could be characterized:

- strength of cryptographic algorithm
  - e.g., RSA, DES, etc., where strength might be measured in terms of the work factor associated with a brute force attack
- Length of cryptographic key
  - characterized by bit-length
- percentage of packets authenticated [14]

  - characterized by percentage of total (e.g., a multimedia environment might tolerate a percentage of data modification or loss)

- Security functions present in destination job-execution environment
  - characterized by operating system or boundary control security policy enforcement mechanisms
- Confidence of policy-enforcement in remote login environment
  - characterized by 3rd-party evaluation
- robustness of authentication mechanism
  - here the range might span weak password, strong password, biometric, and smart cards with on-board display and input interfaces

As one last example, consider a network consisting of various subnets. One of these subnets could be known to be toxic to the interests of the host enterprise, as in a subnet of nodes within a hostile country. Now, the toxic subnet could be identified by ID or by a security rating, and application or enterprise policies could prohibit routing through, execution within, or logon to such a subnet, by specifying allowed or disallowed sets of subnets.

1.3 Goal: Effective Security with regard to QoS Choices

As stated above, network services may allow security ranges. These ranges provide the RMS with additional variables to consider in scheduling and balancing its various requirements. The RMS may allow users and applications to indicate a choice or preference within any of the security ranges. As with other QoS factors, the RMS may modify the level of security service within this range in order to balance other factors, e.g., completion time.

We desire to be able to measure the level of security provided by the RMS in managing the tasks and resources for which it is responsible. Our general approach will be to summarize the level of security service supplied across all scheduled tasks. This metric should give maximum credit to the RMS when it maximizes the security provided to the overall network (i.e., the sum of the network applications). Additionally, we would like to factor user and application security choices into the RMS measurement model so that, if the user asks for, and is provided, less security service than the maximum for a given range, the RMS is not penalized with respect to the metric.

The rest of this paper provides a description of how user and application choices in the context of QoSS can be understood to affect the overall service provided by an RMS. A set of definitions and conceptual framework for reasoning about the QoSS problem is introduced in Section 2. A more formal presentation of choices and restrictions within the context of an RMS is presented in Section 3. Finally, our conclusions and future work are found in Section 4.

2 Network System Model

In order to reason about the defined problem we will first establish a definitional framework.

2.1 Security Resources, Services and Requirements

A network *system* is the infrastructure consisting of the totality of network-accessible resources and security services. A *security service* is a high-level abstract resource providing security functionality such as: authentication, auditing, privacy, integrity, intrusion detection, non-repudiation, and traffic flow confidentiality [3]. A security service typically consumes other low-level system resources such CPU, memory, disk, and network bandwidth. For example, the Common Data Security Architecture (CDSA) [10, 13] describes modules each of which contain specific security mechanisms to provide some of these services.

Each such resource and service may embody security *requirements.* A requirement may *restrict* the availability of a resource to an external entity. Some restrictions might be the typical MAC and DAC requirements, or other security constraints, e.g.: encryption available 9 P.M. to 5 A.M., range of available encryption algorithms, and range of required key lengths.

An implemented security mechanism functions as either a service or a requirement. For example, suppose that a user wishes to access a database within a protected subnet. To access this subnet all packets must be digitally signed using HMAC-SHA. The need to digitally sign all packets entering the subnet is a *requirement* imposed in order to access the database. The user may choose to not access the database because of this imposed requirement. In contrast, if a user has a requirement for data authenticity, he will choose to access the database resource within the protected subnet and will choose to use a local packet authentication *service* in order to transmit packets with the required signatures.

2.2 Variant Security

To be general, we will define that all security requirements have a *range* of permissible behavior. That is, a range may be *unitary*, or degenerate, in which case it represents no choice. Where a range represents a choice, the requirement is termed *security variant.* All system security services are security variant: since they are invoked at the discretion of the user or application, the range is at least binary (i.e, invoked or not invoked). Some requirements are unitary, while others are variant.

2.3 Task Sequences

In the theory of metacomputing, applications may be broken up into subtasks each of which may be executed on different topographical network elements, the results of which are in some way logically joined by the metacomputer [1]. Depending on the metacomputing mechanism used, the topographical structure and the location of specific elements may be more or less transparent to the end user. For the work discussed in this paper, we make the simplifying assumption that a *task* is an application invoked by a user, and each such distributed subtask (if present) is a logically separate *task*. The task utilizes various network system services and resources. The utilization is intermediated by the RMS. Thus, a task is invoked in a sequence:

- the user activates the application through some interface with an application manager (OS, browser, etc.);
- the application is intermediated by the RMS; and
- the RMS submits the application to the system.

We call this the *task invocation sequence:*

$$user \Rightarrow application \Rightarrow RMS \Rightarrow system$$[2]

Security requirements may be established or refined by any or all of: the user, the application, the RMS, and the system. We designate these entities as *security requirement providers.*

As an example of how a requirement can be refined within the task invocation sequence, consider how a typical application offers the user a choice for some service. If the user does not indicate a choice, the application uses some default value. If the user chooses a range, the application invokes itself with a particular value within that range (the application's choices may be managed by a handler or wrapper). Similarly, the RMS may refine the application's choice, for example, to optimize metacomputer performance, load balancing, etc.

In a task invocation sequence, the request is passed from left to right, from a *previous* requirement provider, and to the *next* provider. A security choice for each variant security requirement is logically included with each request step. The choice may be implicit or explicit. For example, if no explicit choice is made, then it may be implicit that the choice is to not limit or modify the security options proffered at that step.

2.4 Security Limits and Choices

Each requirement provider may specify a choice range for each variant requirement in a given task invocation. For example, the user selects a range of 50 - 80% for packet authentication rate. This choice is passed to the next provider (viz., the application) in the sequence. Additionally, each requirement provider may have a *requirement limit range* outside of which it will not accept a request. The limit applies to the request choice from the previous provider, e.g., a given application will not accept a range wider than 60 - 100% from the user. We consider this *limit* to be valid only if the provider enforces it.

[2] Note that it is an implementation detail whether the RMS returns advisory parameters to the application and the application *invokes* the system, or the RMS submits the application with those parameters directly to the system. For simplicity, we assume, here, that the RMS submits the application to the system.

2.5 Range Relationships Inherent in Task Sequences

Table 2 shows the various limits and choices we have identified for security requirement providers.

Table 2: **Security Limits and Choices**

	User	Task	RMS	System
Choice Range	requirement	requirement	requiremnt	Service Level
Limit Range	n/a	requirement	requirement	requirement

Notice that the user does not have an effective limit range, as he has no *previous provider* upon whom to enforce such a range. Also, the system choice range is the level of service ultimately provided by the system in response to the request. This is a unitary range, since there is no *next* provider to whom a choice might be given.

The question arises as to how these ranges relate to each other. We present the following relationships as intuitively inherent in task sequences. However, it is not clear that these relationships are (precisely) necessary or sufficient for that purpose; rather, they are provided to explore the semantics of security ranges in task sequences.

- Each provider's choice must be within its own limit.

  This restriction reflects the natural semantics of choices and limits, in that it is natural to respect one's own limits.

- Each choice must be within the previous choice in the sequence

  This reflects a natural protocol to respect the choice of the previous requirement provider: a requirement provider will try to fulfill the request of a previous provider. For example in a quality of service context, a service provider may accept a request if it can be realized, but it will not proceed with wholly divergent parameters.

- Each choice must be within the next limit in the sequence

  This relation is the consequence of our definition that limits are enforced by their providers. This restriction intuitively means that requests which are out of bounds will be rejected.

- The limits of each provider in a task sequence must all intersect

  This is a consequence of the need for a choice to be within its own limit, and within the next limit, as well as within the previous choice.

2.6 Further Concepts of Operation

Requirement choices and limits may be *unitary* or *nil*. *Unitary* means only one value (i.e., no choice) is passed to the next requirement provider in the task sequence. A *nil* range denotes no restriction is imposed in the task sequence, and would likely denote acceptance of the previous choice range, such that the previous choice is transparently passed through to the next provider.

3 Formal Representation

In this section we present a formal representation of the framework developed in Section 2.

3.1 Goals

The purpose of formalizing the task sequence model introduced above is to *precisely* characterize security choices for support of QoSS in metacomputing environments. Furthermore, we wish to provide *generality* such that the model does not limit designs and implementations of the basic concepts, and we want to provide *consistency* such that the model does not require self-contradictions in derived implementations.

3.2 Range Definitions and Operations

A *range* is a set of elements which defines the possible choices of a variant security requirement. More than just a set, the elements of a range are related, because some are more secure than others. We will use the operator $\geq$ (*dominates*) to partially order the elements of a range with respect to relative security. The dual of the $\geq$ operator is the $\leq$ operator.

These are some example orderings based on such an operator:

- $BlackNet \geq RedNet \geq WWW$
  - Since $BlackNet \geq RedNet \geq WWW$, we see that subsets of a network may be partially ordered by set inclusion.
- % of packets encrypted
  - This is a linear partial ordering based on numeric value.
- $3DES \geq DES \geq caesar\_cipher$
  - The strength of encryption algorithms are ordered by crypto-analytic work factor.

The functions max and min are universal upper and lower bounds on a range:

$$\forall e : element(e \in r \rightarrow max(r) \geq e)$$

$$\forall e : element(e \in r \rightarrow min(r) \leq e)$$

The *enclosure* operator $\gg$ means range r *encloses* range s, as follows:

$$r \gg s \rightarrow max(r) \geq max(s)\ and\ min(s) \geq min(r)$$

The *intersection* operator $\wedge$ means range r and range s intersect:

$$r \wedge s \rightarrow max(r) \geq min(s)\ and\ max(s) \geq min(r)$$

The *sequence* function seq(p) provides the following linear ordering on the security providers (p) of a task sequence:

$$seq(user) < seq(application) < seq(RMS) < seq(system)$$

We also represent the set of security requirement providers:

$$\{u, a, r, s\}$$

This set represents (respectively) the user, application, RMS and system.

To represent the security requirement provider choices and limits discussed above, we introduce the structure **b** in Table 3. **b** consists of two subvectors: one for choice ranges (**b.c**) and one for limit ranges (**b.l**).

Table 3: **Choices and Limits Represented in Structure b**

Entity	Choice	Limit
User	**b.c.u** - user choice range	**b.l.u**- no user limit
Appln	**b.c.a** - application choice range	**b.l.a** - application limit range
RMS	**b.c.r** - RMS choice range	**b.l.r** - RMS limit range
System	**b.c.s** - system response	**b.l.s** - system limit range

3.3 Expression of System Security

Previous work has provided an expression for security requirements in a network environment [3]. Briefly, a security vector S represents the security requirements involving a task executing in a network environment. A security vector component, **S.component**, contains a boolean statement regarding security requirements for a given service or resource.

Examples of security vector components are:

S.a	=	level (user) $\geq$ level(resource)
S.b	=	length of confidentiality encryption key $\geq 64, \leq 256$; *inc* 64
S.c	=	% packets authenticated $\geq 50, \leq 90$, *inc* 10
S.d	=	authentication header transform in {HMAC-MD5, HMAC-SHA}

Each **S.component** has at most one variant requirement. Requirements of a given security service may span several vector components (indicating a service *sub-vector*).

In **S.c**, "inc 10" indicates that the range from 50 through 90 is quantized into increments of 10, viz: 50, 60, 70, 80, 90. Later, we will need to indicate the number of quantized steps in the component; to do this, one more notational element is introduced, $|$ **S.c** $|$. In the above examples, $|$ **S.a** $|= 1$, and $|$ **S.c** $|= 5$.

$$|\ \mathbf{S.c}\ | \quad = \quad \text{number of quanta in } \mathbf{S.c}$$

When $|$ **S.c** $|> 1$, the underlying control program has a $range$ within which it may allow the task to execute with respect to the policy requirement. This range corresponds to the ranges in the structure **b** discussed above, but to which range in **b** does **S.c** correspond? First, we relate a **b** structure to each security vector component **S.c**, as follows:

$$\mathbf{S.c} \models \mathbf{b}$$

We wish to measure the effectiveness of the RMS decision/management strategy as reflected in the ultimate system choices (b.c.s), but we need to provide a measurement "yardstick." The effects of providers who are earlier than the RMS in the task invocation sequence

$$user \Rightarrow application \Rightarrow RMS \Rightarrow system$$

determine the reduced/restricted requirements perceived by the RMS. That is to say, the user and the application may narrow the requirement range such that the requirements are less restrictive[3] than the system maximum, (max(**b.l.s**)) and more restrictive than the system minimum (min(**b.l.s**)), so we will use the reduction immediately before **b.c.r** (i.e., **b.c.a**) as the security metric against which RMS effectiveness is to be measured: the value of the requirement **S.c** is defined to be **S.c** $\models$ **b.c.a**.

3.3.1 Expression of Range Relationships

We can now restate the range relationships described previously, using S and b.

```
$\forall$ $S$:security_vector, c:service_component(
    /* each provider's choice must be within its own limit */
    $\forall$ $p$:provider (
        $S.c \models b.l.p \gg S.c \models b.c.p$)
    & $\forall p_1, p_2$:provider(
        /* each choice must be within the previous choice in the sequence */
        $seq(p_1) < seq(p_2) \rightarrow S.c \models b.c.p_1 \gg S.c \models b.c.p_2$
        /* each choice must be within next limit in the sequence */
        $\& \; seq(p_1) < seq(p_2) \rightarrow S.c \models b.l.p_2 \gg S.c \models b.c.p_1$
```

[3] Here more restrictive means more security.

```
/* all limit ranges intersect */
& $S.c \models b.l.p_1 \wedge S.c \models b.l.p_2$))
```

3.3.2 Expression of Effective Security

To see how effectively the RMS applies variant security to tasks, we need to have an expression for the level or difficulty of security "met" by a task invocation, with respect to the required security range.

From above, $|< range >|$ is the number of quanta in "range."

Let $< response >:< range >$ indicate the ordinal of the quantum in a "range" achieved by a system "response." For example, if a range was from 3 to 12 in increments of 3 then $\mid (3,6,9,12) \mid = 4$. The ordinal of the quantum for 6 is 2, i.e. $6 : (3,6,9,12) = 2$, and the ordinal of the quantum for 9 is 3, i.e. $9 : (3,6,9,12) = 3$.

Then we associate a token $(g.c)$ with each c in S, and define $g.c$ to be the $fraction$ of the required security met by a task invocation:

$$g.c = \frac{< response >:< range >}{|< range >|}$$

We know that

$$requirement = \mathbf{b.c.a} \text{ (security choice of the application)}$$

and

$$met = \mathbf{b.c.s} \text{ (service level provided by system)}$$

So, for example, given **S.c** such that **S.c** $\models$ **b** is defined as follows ...

b.l.s	-system limit range	=	% packets authenticated $\geq 50, \leq 90,$ *inc* 10
b.c.u	-user choice range	=	% packets authenticated $\geq 50, \leq 90$
b.c.a	-appl'n choice range	=	% packets authenticated $\geq 50, \leq 80$
b.c.r	-RMS choice range	=	% packets authenticated $\geq 50, \leq 70$
b.c.s	-system response	=	% packets authenticated $= 70$

(Here, 70 is the third quantum in a range that spans from 50 to 80 in increments of 10.) then we can state:

$$g.c = \frac{b.c.s : b.c.a}{\mid b.c.a \mid} = \frac{3}{4} = 0.75$$

Notice that $g.c = (0 \ or \ 1)$ for invariant components.

With g.c in hand, we introduce a function (A) which averages the tokens of a task [3]:

$$A = \frac{(g_1 + g_2 + .. + g_n)}{n}$$

where n equals the number of components in S. (Optionally, one could add weight coefficients to each g to indicate relative importance with respect to the security policy.)

Since an RMS manages the execution of many tasks over a period of time, we wish to examine its effectiveness with respect to a group of tasks. If A is derived against n tasks, then A_S is an expression for the overall effective security delivered by the RMS to the n tasks:

$$A_S = \frac{\sum_{j=1}^{n} A_j}{n}$$

$0 \leq A_S \leq 1$, where 1 indicates the maximum scheduling effectiveness.

4 Conclusion

We have presented a framework for analysis of security with respect to Quality of Security Service, user security choices, and the "task invocation sequence" used in a metacomputer scheduling mechanism (e.g., an "RMS"). This framework characterizes the relationships of security requirements of users, applications, RMSs, as well as the underlying security resources and services. We then presented a formalization of the framework, and extended it to include an expression for the efficiency of an RMS with respect to security, which tends to the maximum as security "met" approaches the maximum of the application's choice.

4.1 Future Work

We are currently working on several aspects of security regarding the characterization of RMS security, security choices, and benefit functions. We are also working on incorporating *variant* security costing techniques into a research prototype of an RMS [9, 3, 17].

We would like to understand the completeness of the range relationships identified for task invocation sequences.

We are working to better understand the nature of contradictions which might exist within or between security vector components. For example, does there exist a normal form of S whose derivation discovers or eliminates contradictions? One such contradiction involves a security component whose provider limits (e.g., application limit and RMS limit) do not intersect; another involves two interdependent components of a security vector (e.g., a communication authentication service which utilizes a digital integrity service), whose limits for a given requirement (e.g., the required ranges for encryption key lengths) do not intersect.

We would like to incorporate into the benefit function the effects of parallel and redundant security mechanisms [7].

References

[1] Ammar Alhusaini, Viktor K. Prasanna, and C.S. Raghavendra. A Unified Resource Scheduling Framework for Heterogeneous Computing Environments. In *Proceedings of the Heterogeneous Computing Workshop*, pages 156–165, San Juan, PR, April 1999. IEEE Computer Society Press.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173, Oakland, CA, May 1996. IEEE Computer Society Press.

[3] blind. This reference is blinded for peer review, January 3000.

[4] Tracy D. Braun, Muthucumaru Maheswaran, Howard Jay Siegel, Noah Beck, Ladislau Boloni, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, , and Bin Yao. A Taxonomy for Describing Matching and Scheduling Heuristic for Mixed-Machine Heterogeneous Computing Systems. In *Workshop on Advances in Parallel and Distributed Systems (in proceedings of the IEEE Symposium on Reliable Distributed Systems)*, pages 330–335, West Lafayette, IN, October 1998.

[5] Paul Carff. When is a simple model adequate for use in scheduling in mshn? Master's thesis, Naval Postgraduate School, Monterey, CA, March 1999.

[6] S. Chatterjee, B. Sabata, and J. Sydir. Erdos qos architecture. Technical Report ITAD-1667-TR-98-075, SRI International, Menlo Park, CA, May 1998.

[7] Saurav Chatterjee and Michael Brown. A graceful adaptation scheme for secure multimedia traffic. Technical Report ITAD-1667-PA-99-008, SRI International, Menlo Park, CA, 1999.

[8] I. Foster and C. Kesselman. Globus: A metacomputing ingrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[9] Debra Hensgen, Taylor Kidd, David St. John, Matthew C. Schnaidt, J.J. Siegel, Tracy Braun, Jong-Kook Kim, Shoukat Ali, Cynthia Irvine, Tim Levin, Victor Prasanna, Prashanth Bhat, Richard Freund, and Mike Gherrity. An Overview of the Management System for Heterogeneous Networks (MSHN). In *8th Workshop on Heterogeneous Computing systems*, pages 184–198, San Juan, PR, April 1999.

[10] Intel. *Common Data Security Architecture Specification*. Intel Corporation, Santa Clara, CA, release 1.2 edition, February 1998. http://developer.intel.com/ial/security/.

[11] Jong-Kook Kim, Debra Hensgen, Taylor Kidd, H. J. Siegel, David St. John, Cynthia Irvine, Timothy Levin, N.W. Porter, Victor Prasanna, and Richard Freund. A QoS Performance Measure Framework for Distributed Heterogeneous Networks. In *Proceedings of the 8th Euromicro Workshop on Parallel and distributed Processing*, pages 18–27, Rhodes, Greece, 2000.

[12] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the Heterogeneous Computing Workshop*, pages 30–44, San Juan, PR, April 1999. IEEE Computer Society.

[13] Richard Sargent. *CDSA Explained: An Indispensable Guide to Common Data Security Architecture*. The Open Group, Reading, Berkshire, UK, 1998.

[14] P. Schneck and K. Schwan. Dynamic authentication for high performance networked applications. Technical Report GIT-CC-98-08, Georgia Institute of Technology, College of Computing, Atlanta, GA, 1998.

[15] L. Smarr and C. Catlett. Metacomputing. *Communications of the A.C.M.*, June 1992.

[16] N. Vendatasubramanian and K. Narstedt. An integrated metric for video qos. In *A.C.M. International Multimedia Conference*, Seattle, WA, November 1997.

[17] Lonnie Welch, Michael W. Masters, Leslie Madden, David Marlow, Philip Irey, Paul Werme, and Behrooz Shirazi. A Distributed System Reference Architecture for Adaptive QoS and Resource Management. In Jose Rolim, editor, *Proceedings of 11th IPPS/SPDP'99 Workshops*, pages 1316–1326, Berlin, April 1999. Springer.